

47

C'EST BIENTOT L'AUTOMNE, LES FEUILLES (DE LISTING...?) MÛRISSENT. JUILLET 1988

DE L'IDÉE

À LA RÉALISATION...

EDITORIAL

J'avais comme idée de couverture un dessin représentant une faucille et un marteau, la faucille avec un bouchon planté à l'extrémité et le marteau enveloppé dans une serviette. Le slogan l'accompagnant aurait été du style" PC: y-a-t-il du mou dans le dogme?". Tout ceci bien entendu à prendre, dans mon esprit, au second degré, PC signifiant Personnal Computer (qu'alliez-vous croire?). Mais en ces temps d'intégrisme religieux à la vocation incendiaire, je me suis retenu, ne voulant choquer aucune sensibilité. Mais quand même, il y a du mou dans le dogme IBM. Sitôt les PS sortis, nous attendions la première dissidence made in Taiwan nous expédiant par boat-computer entiers des clônes PS à quart de prix. Que nennni, Big Machin a breveté et protégé son bus: personne ne monte sans payer. Alors les cloneurs ont décidé de pondre leur bus à eux, volant à droite, doubles soupapes, culbuteurs chromés, pare-choc lubrifié, et tutti quanti. Big Machin fait la tête, car les cloneurs vendent des PS dont les logiciels sont compatibles, mais pas les extensions. Mais qui achèterait une extension Big Machin contre trois extensions Made in Korea-Taïwan-Hong Kong (rayer -en jaune- celui qui ne vous fournit pas...). Va-t-il y avoir de l'avortement sur chaine chez Big Machin, comme il l'avait déja fait avec son JrPC que j'ai vu (une fois...) mais jamais touché.

SOMMAIRE

Nos coordonnées: ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS
Tel président: 43.40.96.53
Tel secrétaire: 49.85.63.67
TELETEL 3: 3615 SAM*JEDI (Forum, BALs, Téléchargement)

FORTH

GESTION DE FENETRES PAR PILE

Système: TURBO-Forth 83-Standard
Adaptabilité: néant
Diffusion: téléchargement 3615 SAM*JEDI
et module M6

Ce programme a été écrit pour une carte Hercules, je pense qu'il y a peu de modifications à apporter pour l'adapter à une carte CGA. Ne possèdant pas ce type de carte, je laisse à vos bons soins les commentaires nécessaires (si votre temps le permet).

Ce programme de fenêtrage permet de stocker les données en dehors du segment Forth. Les fenêtres sont décomposées en deux classes, les fenêtres permettant d'afficher des menus et celles permettant d'afficher du texte dans une portion définie de l'écran. L'ensemble est architecturé autour d'une pile dont la structure convient parfaitement à la gestion des ouvertures et fermetures de fenêtres, une pile supplémentaire à été ajoutée afin d'assurer la gestion des barres de menus.

Comme les données sont stockées en dehors du segment forth, il sera nécessaire de les charger avant de lancer un programme. Deux solutions sont possibles, soit on sauvegarde l'image mémoire ou sont stockées les données, soit on charge un fichier qui recomposera les fenêtres. La deuxième solution a été choisie car elle a le mérite de ne pas trop encombrer la mémoire de masse.

Le fichier FWIND.FTH est composé des utilitaires nécessaires à la gestion des fenêtres. FWMENU.FTH est un exemple avec son fichier de recouvrement FWMENU.OVL. Dans cet exemple lorsque dans le menu DOS, on donne des directives erronées, Turbo-Forth envoie un message d'erreur par l'intermédiaire de ?DOS-ERR composé lui même de la procédure ABORT"... Ceci a pour effet d'interrompre le programme et de "coincer" l'utilisateur dans la fenêtre en cours. On peut en sortir en effectuant autant de WCLOSE qu'il y a de fenêtres (d'ailleurs ne serait-il pas préférable dans turbo-forth de vectoriser ?DOS-ERR afin de laisser à chacun le choix de la gestion des erreurs DOS, ou alors de proposer un mot permettant de transformer des procédures classiques en vecteurs ?).

Les extensions qu'il semble possible d'ajouter concernent des mots de manipulation de la pile des fenêtres, un SWAP pourrait être intéressant pour basculer l'affichage d'une fenêtre de texte à l'autre.

On peut difficilement faire un bon programme de fenêtrage en restant aussi haut dans le dictionnaire. Si on descend la procédure WPAGE au niveau de EMIT en définissant l'écran comme la première fenêtre ouverte, il devient possible d'exécuter des scripts à l'intérieur d'une fenêtre spécifiée par la programme appelant. Par exemple le mot PROGRAM lancera une application Forth avec dans sa ligne de commande les coordonnées de la fenêtre d'affichage (il est possible de créer cette procédure à la suite des utilitaires déjà présentés, mais cette solution n'est pas esthétique).

Suite à ces commentaires, j'aimerai proposer quelques remarques:

- Je ne suis pas informaticien de formation, et je possède des lacunes quand à la compréhension de certains concepts nomades véhiculés par Turbo-Forth.

Si le multi-tâche, la gestion des vocabulaires, la méta-interprétation (bravo!) passent bien, la méta-compilation est un problème apparemment imperméable qui ne trouve aucun commentaire conséquent tant dans Jedi que dans d'autres revues.

Je rêve depuis quelques temps de bricoler un Forth spécifique à mes besoins (traitement d'images), mais je suis incapable de bien maitriser les techniques de méta-compilation. Alors j'adresse une requête, ne serait il pas possible qu'un méta-barbu de la compilation puisse glisser quelques mots salvateurs sur le procédé et les possibilités de la chose? (Si un tel article apparait, dans les mois suivants, je vous emprisonne un script forth dans une fenêtre).

- L'utilisation que je fais de Forth m'amène à construire des structures arborescentes et des graphes. Ce n'est apparemment pas un pôle d'intéret pour les adhérents de Jedi, mais si des personnes sont intéressées, c'est avec plaisir que je partagerai certaines procédures arboricoles.

- Il est certain que Jedi et Turbo-Forth constituent le top-level dans le monde Forth, mais serait-il possible de donner les coordonnées des revues qui traitent de Forth, ainsi qu'un descriptif même sommaire des langages dérivés de Forth.

FICHIER: FWIND.FTH

```
\    ***********************************
\    * Allocation et libération mémoire *
\    ***********************************
HEX
\ Libération de la mémoire allouée par un fichier .COM
CODE LIB-COM
        CS AX MOV AX ES MOV
        1000 # BX MOV 4A # AH MOV 21 INT
        NEXT
END-CODE
\ Réservation mémoire pour le stockage des fenetres
CODE MALLOC
        BX POP 48 # AH MOV 21 INT
        1PUSH
END-CODE
\ Libération de la mémoire réservée
CODE RELEASE
        ES POP 49 # AH MOV 21 INT
        NEXT
END-CODE
VARIABLE WSEGMENT          \ Adresse du segment de stockage
                           \ des fenetres.
B000 CONSTANT RAMECR       \ Carte Hercules.
DECIMAL

 \ Limitations à 20 fenetres ouvertes simultanéments
 \ Pour augmenter il faut modifier ISPACE et WSPACE
 \ Pile des barres de menu
 20 ALLOT 0 ,              \ Pile des barres de menu
 HERE 1- CONSTANT I0
 140 ALLOT 0 ,             \ Pile fenetre
 HERE 1- CONSTANT W0
 80 CONSTANT C\L

 VARIABLE *IP                  \ Pointeur pile indice
 VARIABLE *ATTRIBUT            \ Attribut vidéo
 VARIABLE *LG                  \ Longueur de la fenetre
 VARIABLE *HT                  \ Hauteur de la fenetre
 VARIABLE *COL                 \ Coin supérieur gauche
 VARIABLE *LIG                 \ idem
 VARIABLE *WPTR                \ Pointeur sur offset
                               \ dans Wsegment
 VARIABLE *IND                 \ Numéro du menu choisi
 VARIABLE *WP                  \ Pointeur pile fenetre
 VARIABLE WFLAG                \ Indicateur de sortie
 VARIABLE WLG                  \ Lig Debut d'une fenetre
                               \ texte
 VARIABLE WHT                  \ Col Debut      "
 VARIABLE WMAXLG               \ Lig fin        "
 VARIABLE WMAXHT               \ Col fin        "
 0 CONSTANT WM                 \ Indicateur menu
 1 CONSTANT WT                 \ Indicateur fenetre
                               \  texte

 2 *ATTRIBUT !
 1 *IND !  *WPTR OFF  W0 *WP !   I0 *IP !

 HEX
```

```
\ Scrolling d'une fenetre   n SCROLL  n nombre de lignes
CODE SCROLL
     AX POP 2 # BH MOV
     WMAXLG #) DL MOV
     WMAXHT #) DH MOV
     *COL #) CL MOV
     1 # CL ADD
     *LIG #) CH MOV
     1 # CH ADD
     6 # AH MOV  10 INT NEXT
END-CODE
\ Transfert de l'écran vers la zone de stockage
CODE S->W                    \ origECR destWSEG  len
        CX POP DI POP BX POP BX SHL
        WSEGMENT #) DX MOV
        RAMECR # AX MOV
        AX DS MOV DX ES MOV
        SI BX XCHG
        HERE  BYTE MOVS AL LODS LOOP
        CS AX MOV AX ES MOV
        AX DS MOV SI BX XCHG
        NEXT
END-CODE
\ Transfert de la zone de stockage vers l'écran
CODE W->S                  \ oriWSEG destSCR  len
        CX POP DI POP DI SHL
        BX POP CX PUSH
        *ATTRIBUT #) CX MOV
        WSEGMENT #) AX MOV
        RAMECR # DX MOV
        AX DS MOV  DX ES MOV
        CX AX MOV CX POP SI BX XCHG
        HERE  BYTE MOVS AL STOS LOOP
        CS AX MOV AX ES MOV
        AX DS MOV SI BX XCHG
        NEXT
END-CODE
\ Code clavier de la touche enfoncée.
CODE CKEY
     00 # AX MOV 16 INT
     AH AL MOV AH AH XOR 1PUSH
END-CODE
DECIMAL

\  *****************************************************
\  *   PROCEDURES FORTH DE MANIPULATION DE FENETRES    *
\  *              DE MENUS ET DE TEXTES                *
\  *****************************************************
\  * long larg WINDOW (nomFenetre)                     *
\  * EPAIS FIN ... aspect du cadre                     *
\  * WCAD! tracé du cadre                              *
\  * WT Spécification d'une fenetre de texte           *
\  * WM spécification d'une fenetre de menu            *
\  * WOPEN ouverture d'une fenetre                     *
\  * WCLOSE fermeture de la dernière fenetre ouverte   *
\  * WMOVE Déplace la dernière fenetre ouverte         *
\  * WDISPLAY affiche une fenetre sans agir sue la pile *
\  * WMENU active la barre de menu                     *
\  *  *Curseur haut et bas deplacement barre de menu   *
\  *  *Return valide un choix                          *
\  *  *Esc sortie du menu                              *
\  *****************************************************

\ Taille Minimum : 3 3 WINDOW (NOM).
: WINDOW
        CREATE                       \ lg ht WINDOW Wxxx
           DUP C, SWAP DUP C,
           *WPTR @ DUP ,     \ pointeur sur offset WSEGMENT
           -ROT * 2*         \ taille = fenetre+ombre
           2DUP WSEGMENT @ -ROT BL LFILL
           + *WPTR !
        DOES>
           DUP C@
           SWAP DUP 1+ C@
           SWAP 2+ @
           -ROT ;

: CADRE
       CREATE
            C, C, C, C, C, C, C,
       DOES>
            SWAP + C@ ;
\ 7htg 6htd 5bg 4bd 3horh 2horb 1ver

  219  219  219 219 223   220   219 CADRE C1  \ EPAIS
  218  191  192 217 196   196   179 CADRE C2  \ FIN
  201  187  200 188 205   205   186 CADRE C3  \ DOUBLE
  214  183  211 189 196   196   186 CADRE C4
                                      \ Fin en haut
                                      \ Double en vertical
  213  184  212 190 205   205   179 CADRE C5
                                      \ Double en haut
                                      \ Fin en vertical
DEFER CAD
: EPAIS  ['] C1 IS CAD ;
: FIN    ['] C2 IS CAD ; FIN \ Par défaut
: DOUBLE ['] C3 IS CAD ;
: MIX1   ['] C4 IS CAD ;
: MIX2   ['] C5 IS CAD ;
: WMET  WSEGMENT @ SWAP ;
: WCAD!                                \ (NOM) WCAD!
   DUP *LG ! SWAP DUP *HT !
   * OVER +
   3 CAD OVER 1- WMET LC!
   4 CAD OVER *LG @ - TUCK WMET LC!
   WMET 1+ *LG @ 2- 1 CAD LFILL
   *LG @ 1+ - SWAP
   6 CAD OVER WMET LC!
   5 CAD OVER WMET *LG @ 1- + LC!
   DUP 1+ WMET *LG @ 2- 2 CAD LFILL
   *LG @ + DO
          I 0 CAD OVER WMET LC!
          *LG @ 1- + 0 CAD WSEGMENT @ ROT LC!
        *LG @ +LOOP ;
: W$!                        \ " xxx" pos (NOM) w$!
   DUP *LG ! NIP ROT
   * + 1+ WMET ROT
   *LG @ 2- MIN
   >R >R DSEGMENT -ROT R> R> LCMOVE ;
: WINIT                               \ adr ht lg lig col
   *COL ! *LIG !
   DUP *LG ! SWAP DUP *HT ! * ;
VARIABLE INCLIG
: (WOPEN)                             \ adr ht lg lig col
   WINIT *LIG @ INCLIG !
   2DUP 2* + -ROT TUCK + ROT SWAP
   DO
     INCLIG @ C\L * *COL @ + DUP I *LG @ S->W
     SWAP I OVER - ROT *LG @ W->S
     INCLIG 1+!
   *LG @ +LOOP
   DROP ;
: (WCLOSE)
   WINIT
   2DUP 2* + -ROT  +
   DO
     I *LIG @ C\L * *COL @ + *LG @ W->S
     1 *LIG +!
   *LG @ +LOOP ;
: WP@ *WP @ ;
: WPUSH                         \ Empile une fenetre
   WP@ 7 - W0 140 - < ABORT" Wpile pleine !"
   WP@ C!  WP@ 1- C!  WP@ 2- C!
   WP@ 3 - C!  WP@ 4 - C! WP@ 6 - !
   -7 *WP +! ;
: WPOP                          \ Depile une fenetre
   WP@ W0 = ABORT" Wpile vide !"
   WP@ 1+ @  WP@ 3 + C@  WP@ 4 + C@
   WP@ 5 + C@  WP@ 6 + C@  WP@ 7 + C@
   7 *WP +! ;
: WREP                          \ Duplique sur la pile
   WP@ W0 = ABORT" Wpile vide !"
   WP@ 1+ @  WP@ 3 + C@  WP@ 4 + C@
   WP@ 5 + C@  WP@ 6 + C@  WP@ 7 + C@ ;
: ?WSTACK
   WP@ W0 < ;
: IP@ *IP @ ;
: IPUSH                          \ Empile la position de
   IP@ 1- I0 20 - < ABORT" Pile Indice Pleine"
                                   \  la barre menu
   *IND @ IP@ C! -1 *IP +! ;  IPUSH
: IPOP                           \ Dépile
   IP@ I0 = ABORT" Pile Indice Vide"
   IP@ 1+ C@ *IND ! 1 *IP +! ;

\ Gestion de la barre de menu
```

```
: WSELECT                          \ adr ind
  TUCK *LG @ * + 1+ SWAP
  *LIG @ + C\L * *COL @ + 1+
  *LG @ 2- W->S ;
: SUIV                             \ adr
  2 *ATTRIBUT ! DUP *IND @ WSELECT
  120 *ATTRIBUT ! 1 *IND +!
  *IND @ 1+ *HT @ 1- > IF 1 *IND ! THEN
  DUP *IND @ WSELECT
  2 *ATTRIBUT ! ;
: PREC                             \ adr
  2 *ATTRIBUT ! DUP *IND @ WSELECT
  120 *ATTRIBUT ! -1 *IND +!
  *IND @ 0= IF *HT @ 2- *IND ! THEN
  DUP *IND @ WSELECT
  2 *ATTRIBUT ! ;
\ GESTION DES FENETRES TEXTES
\ Ajustement de la position du curseur
: WPAGE
  DUP 13 = IF WMAXHT @ WHT @ >
             IF *COL @ 1+ WLG ! WHT 1+!
             ELSE 1 SCROLL *COL @ 1+ WLG ! WMAXHT @ WHT !
             THEN
           THEN
  DUP  8 = IF WLG @ *COL @ 1+ >
             IF WLG 1-!
             ELSE *COL @ 1+ WLG !
             THEN
           THEN
    32 >= IF WMAXLG @ WLG @ >
            IF   WLG 1+!
            ELSE WMAXHT @ WHT @ >
               IF  *COL @ 1+ WLG ! WHT 1+!
               ELSE 1 SCROLL *COL @ 1+ WLG !
                    WMAXHT @ WHT !
               THEN
            THEN
          THEN ;
: TEXTINIT
  *COL @ 1+ WLG !
  *LIG @ 1+ WHT !
  *COL @ *LG @ 2- + WMAXLG !
  *LIG @ *HT @ 2- + WMAXHT !
  WLG @ WHT @ (AT) 2DROP ;
: (WCONS)
  DUP (EMIT) WPAGE WLG @ WHT @ (AT) 2DROP ;
: (WDARK)
  0 SCROLL 0 0 AT ;
\ En cas de valeurs supérieures aux dimensions maxi,
\ L'affichage se fera en dehors de la fenêtre avec pour
\ origine écran le coin supérieur gauche de la fenêtre.
: (WAT)
  *LIG @ 1+ + WHT ! *COL @ 1+ + WLG !
  WLG @ WHT @ (AT) ;
: CURSON
  1 0 (FRAME) ;
: CURSOFF
  15 0 (FRAME) ;
: LIMECR                   \ Environnement fenetre
  ['] (WCONS) IS EMIT
  ['] (WDARK) IS DARK
  ['] (WAT) IS AT ;
: NORMECR                  \ Environnement ecran
  ['] (EMIT) IS EMIT
  ['] (MODE) IS DARK
  ['] (AT) IS AT ;

\ PROCEDURES D'AFFICHAGES
: WOPEN                    \ <NOM> lig col ind WOPEN
  WPUSH WREP
   IF  CURSON (WOPEN)
       TEXTINIT LIMECR
   ELSE
       CURSOFF (WOPEN)
   THEN ;
: WCLOSE        \ ferme la dernière fenetre ouverte
  WPOP DROP (WCLOSE)
  ?WSTACK IF
          WREP IF
                 CURSON WINIT 2DROP
                 TEXTINIT LIMECR
               ELSE
                 CURSOFF WINIT 2DROP
                 NORMECR
               THEN
          ELSE
             CURSON NORMECR
          THEN ;
\ Déplace uniquement une fenetre déjà ouverte
: WMOVE         \ <NOM> lig col ind WMOVE
  WCLOSE WOPEN ;
\ Affiche la fenetre présente au sommet de
\ la pile des fenetres.
: WDISPLAY
  WREP DROP WINIT *LIG @ INCLIG !
  OVER + SWAP
  DO I INCLIG @ C\L * *COL @ +
     *LG @ W->S INCLIG 1+! *LG @
  +LOOP ;
: WMENU
  WFLAG OFF
  WREP DROP WINIT DROP SUIV PREC
  BEGIN
   CKEY CASE
         80 OF SUIV 0 ENDOF
         72 OF PREC 0 ENDOF
         28 OF DROP *IND @ 1 ENDOF
         01 OF DROP WFLAG ON WCLOSE 0 1 ENDOF
         DUP OF 0 ENDOF
       ENDCASE
  UNTIL ;
EOF
```

```
LISTING:             FWMENU.FTH

 LIB-COM
 2048 MALLOC WSEGMENT !
 *WPTR OFF
 31 6  WINDOW W1      \ MENU GENERAL
 31 5  WINDOW WECS
  \ Edition Compilation Système SaveCore
 10 8 WINDOW W2       \ GESTION DOS
 17 04 WINDOW W3      \ OPTION DU SYSTEME
 40 05 WINDOW W4      \ REQUETES
 78 12 WINDOW W5      \ AFFICHAGE DES DEMANDES: WORDS DIR..

 FIN W1 WCAD! W2 WCAD! W3 WCAD! WECS WCAD!
 EPAIS W4 WCAD! W5 WCAD!
 " Edition/Compilation " 1 W1 W$!
 " Gestion DOS " 2 W1 W$!
 " Exécution d'un mot Forth" 3 W1 W$!
 " Retour au DOS" 4 W1 W$!
 " Edition d'un fichier " 1 WECS W$!
 " Compilation d'un fichier " 2 WECS W$!
 " Options du système " 3 WECS W$!
 5 5 WINDOW W6 FIN W6 WCAD!
 " A:" 1 W6 W$!
 " B:" 2 W6 W$!
 " C:" 3 W6 W$!
 " A: B: C:" 1 W2 W$!
 " DIR " 2 W2 W$!
 " CHDIR " 3 W2 W$!
 " DELETE " 4 W2 W$!
 " MKDIR " 5 W2 W$!
 " RMDIR " 6 W2 W$!
 " Attributs ON " 1 W3 W$!
 " Echo ON" 2 W3 W$!
: WSYSTEM
  W3 4 60 WM WOPEN
  BEGIN
   WMENU CASE
      1 OF IPUSH
           ATTRIBUTS @
           IF ATTRIBUTS OFF
              " Attributs OFF" 1 W3 W$!
           ELSE ATTRIBUTS ON
              " Attributs ON " 1 W3 W$! THEN
           IPOP ENDOF
      2 OF IPUSH
           ECHO @
           IF ECHO OFF " Echo OFF" 2 W3 W$!
           ELSE ECHO ON " Echo ON " 2 W3 W$! THEN
           IPOP ENDOF
  ENDCASE
  WDISPLAY
```

```
  WFLAG @ UNTIL WFLAG OFF ;
30 STRING WW$
40 STRING WVAR$
: WKEY
  CR ." Appuyer sur une touche ..." KEY DROP ;
: WFCT
  WVAR$ $! 1 1 AT WVAR$ TYPE
  WVAR$ NIP 2+ 1 AT WW$ INPUT$ ;
: WEXE
  WW$ WVAR$ APPEND$
  WVAR$ $EXECUTE ;
: WDRV
  W6 4 40 WM WOPEN
     BEGIN WMENU CASE
     1 OF " A:" WW$ $! WW$ $EXECUTE ENDOF
     2 OF " B:" WW$ $! WW$ $EXECUTE ENDOF
     3 OF " C:" WW$ $! WW$ $EXECUTE ENDOF
     ENDCASE
     WFLAG @ UNTIL WFLAG OFF ;
: WDOS
  W2 3 31 WM WOPEN
  BEGIN
     WMENU CASE
     1 OF IPUSH WDRV IPOP ENDOF
     2 OF IPUSH W4 20 20 WT WOPEN DARK
          ."   Spécifier le masque..."
          " DIR " WFCT W5 10 0 WT WOPEN WEXE WKEY
          WCLOSE WCLOSE IPOP ENDOF
     3 OF IPUSH W4 20 20 WT WOPEN DARK
          ."  Changement de Directory."
          " CHDIR " WFCT WEXE WCLOSE IPOP ENDOF
     4 OF IPUSH W4 20 20 WT WOPEN DARK
          ."  Détruit un fichier."
          " DEL " WFCT WEXE WCLOSE IPOP ENDOF
     5 OF IPUSH W4 20 20 WT WOPEN DARK
          ." Création d'un Directory."
          " MKDIR " WFCT WEXE WCLOSE IPOP ENDOF
     6 OF IPUSH W4 20 20 WT WOPEN DARK
          ." Enlève un Directory."
          " RMDIR " WFCT WEXE WCLOSE IPOP ENDOF
     ENDCASE
  WFLAG @ UNTIL
  WFLAG OFF ;
: WEDITCOMP
  WECS 2 30 WM WOPEN
  BEGIN
     WMENU CASE
     1 OF IPUSH WCLOSE WCLOSE DARK EDIT
          W1 1 1 WM WOPEN WECS 2 30 WM WOPEN
          IPOP ENDOF
     2 OF IPUSH W4 20 20 WT WOPEN DARK
          ."  Entrez le nom du fichier."
          " INCLUDE " WFCT WEXE WCLOSE IPOP ENDOF
     3 OF IPUSH WSYSTEM IPOP ENDOF
     ENDCASE
  WFLAG @ UNTIL WFLAG OFF ;
: WGENERAL
  W1 1 1 WM WOPEN
  BEGIN
     WMENU CASE
          1 OF IPUSH WEDITCOMP IPOP ENDOF
          2 OF IPUSH WDOS IPOP ENDOF
          3 OF IPUSH W4 20 20 WT WOPEN DARK
             ." Entrer le mot."
                "  " WFCT W5 10 0 WT WOPEN WEXE WKEY WCLOSE
                WCLOSE IPOP ENDOF
          4 OF WCLOSE WSEGMENT @ RELEASE BYE ENDOF
     ENDCASE
  WFLAG @ UNTIL
  WFLAG OFF ;

EOF
SAVE-SYSTEM <NOM>.COM
Puis pour relancer
<NOM>.COM INCLUDE MENU.OVL WGENERAL
```

```
FICHIER:                FWMENU.OVL

LIB-COM
2048 MALLOC WSEGMENT !
WSEGMENT @ 0 2048 16 * BL LFILL
FIN
W1 WCAD! W2 WCAD! W3 WCAD! WECS WCAD!
DOUBLE
W4 WCAD! W5 WCAD!

" Edition/Compilation          " 1 W1 W$!
" Gestion DOS                  " 2 W1 W$!
" Exécution d'un mot Forth     " 3 W1 W$!
" Retour au DOS                " 4 W1 W$!
" Edition d'un fichier          " 1 WECS W$!
" Compilation d'un fichier      " 2 WECS W$!
" Options du système            " 3 WECS W$!
FIN W6 WCAD!
" A: " 1 W6 W$!
" B: " 2 W6 W$!
" C: " 3 W6 W$!
" A: B: C: " 1 W2 W$!
" DIR      " 2 W2 W$!
" CHDIR    " 3 W2 W$!
" DELETE   " 4 W2 W$!
" MKDIR    " 5 W2 W$!
" RMDIR    " 6 W2 W$!
" Attributs ON   " 1 W3 W$!
" Echo ON        " 2 W3 W$!
```

TELECHARGEMENT SUR ATARI ST.

par Daniel FLIPO

Système : VolksFORTH 83-Standard ATARI ST
Diffusion: 3615 SAM*JEDI dès la mise en place de l'option de téléchargement ATARI.
Copie disquette, demander à 3615 SAM*JEDI Bal FLIPO

Commentaire du Secrétaire:
Décidément, je vais de surprise en étonnement. Il y a un an, TURBO-Forth sortait à peine des limbes. Et maintenant, il est exploité par plus de trois cents programmeurs et des applications ambitieuses remplissent les modules d'extension et le téléchargement sur 3615. Dans ce domaine, nous avons préféré le transfert ASCII, ce qui est une solution peut-être discutable, mais certainement plus universelle que le codage dédié à un seul système (PC et compatibles en l'occurence).

Notre clairvoyance porte ses fruits, car grâce à la persévérence de D. FLIPO, les systèmes ATARI peuvent accéder à toute la bibliothèque TURBO-Forth PC. Bien entendu, il n'est pas question de pouvoir exécuter le code transféré sans aménagement, mais de nombreuses heures de recopie manuelle de listing sont ainsi épargnées.

Dans un article du présent numéro, j'expliquerai comment rendre un même source polyglotte, mais également multi-système, ceci de manière totalement transparente pour l'utilisateur. Je laisse maintenant la plume à Mr FLIPO.

Si l'on veut que notre serveur JEDI remplisse pleinement son rôle, il me paraît utile que:

1) des programmes en Volksforth-83 soient disponibles sur le serveur aux côtés de ceux en TURBO-Forth.

2) les Ataristes comme les adeptes de BIG BLUE puissent utiliser (après adaptation souvent...) les sources disponibles soit en TURBO, soit en VF-83.

Exemple: téléchargement par un Atariste du pack virgule flottante FPACK.FTH écrit en Turboforth. L'adaptation nécessaire est décrite dans l'article "Virgule flottante sur ATARI ST".

Rappelons que pour télécharger vous devez disposer:

- d'un minitel (ou d'un modem...)
- d'un cable de liaison minitel-prise RS232 de l'ordinateur (des schémas de ce cable ont été publiés un peu partout : ST-MAGAZINE N°4 par exemple, et un cable

tout fait peut être acheté dans les bonnes boutiques ou par correspondance à Pressimage cf ST-MAGAZINE).

- d'un logiciel de communication intégrant si possible le protocole KERMIT pour la détection/correction des erreurs de transmission.

Hélas, je n'en connais pas, mais ST-COMM (75 Francs chez Pressimage) vous permettra de télécharger les textes ASCII (Turbo et VolksFORTH) sans problème. Pour lire le Forum ou votre BAL, ST-COMM en mode ASCII convient aussi. Bien sûr, EMULCOM est plus agréable, mais vu son prix on peut s'en passer... EMULCOM ne PERMET PAS LE TELECHARGEMENT MEME DE TEXTES ASCII (il perd un tas de caractères en route !!!). Je compte écrire un logiciel qui pallie aux lacunes des deux en VolksFORTH-83. Quand il sera opérationnel, vous l'aurez dans JEDI avec le source bien sûr !.

Le principal problème est que les sources en Turbo-Forth sont des textes ascii dont les lignes sont terminées par un CR LF, tandis que les sources en VF-83 (comme dans tout Forth classique...) sont faits de blocs de 1024 octets (soit exactement 2 secteurs sur disquette) sans CR ni LF.

Dès lors deux approches sont possibles :

- ou bien on développe en VF-83 un INCLUDE "spécial" qui permette de compiler des textes ascii comme le fait Turbo-Forth,
- ou bien on propose une conversion dans les deux sens ascii -> blocs et blocs -> ascii.

J'ai choisi la seconde solution car les sources en Turbo-Forth ne seront pratiquement jamais compilables sans modification sur une autre machine qu'un compatible IBM (mots en assembleur, différences de vocabulaire, accès au DOS...). Il faudrait donc avant compilation les retravailler sur un traitement de texte extérieur à VF-83. L'éditeur intégré étant excellent, il m'a paru préférable de faire la conversion en blocs puis les adaptations nécessaires sous l'éditeur du VF-83. De plus, il ne me parait pas acceptable d'avoir dans VF-83 deux types de sources (ASCII et BLOCS) incompatibles entre eux ....

Je vous propose donc un fichier CONV BLK.SCR écrit en VF-83 où sont définis les deux mots FTH>BLK (conversion ascii -> blocs) et BLK>FTH pour la conversion inverse.

Quelques remarques :

1) Le mot FTH>BLK arrête la conversion dès qu'il rencontre le mot EOF qui termine le code source. Les commentaires qui suivent éventuellement ne me semblent pas devoir être intégrés dans des blocs. Si vous souhaitez les conserver, récupérez les donc avec un traitement de texte pour en faire un fichier .ASC par exemple. Si vous tenez à convertir tout le fichier, il suffit de supprimer du mot blk format (écran 5) les trois premières lignes suivant le ELSE ... Le test de présence du mot EOF n'a lieu qu'à chaque début de ligne, afin qu'un mot contenant les trois lettres EOF n'arrête inopinément la conversion...

2) Je propose que les sources écrits en VF-83 sur ATARI et destinés à être publiés dans JEDI soient convertis en texte ascii grâce au mot BLK>FTH puis envoyés sur une disquette 3 pouces 1/2 double face (720 ko) MS/DOS. Pour obtenir avec un ATARI une disquette au format MS/DOS, on pourra utiliser le programme QFORMAT.ACC dont le source paraitra également dans JEDI ou, convertir une disquette ATARI en disquette MS/DOS grâce au mot >IBM décrit dans l'article "Rendez vos disquettes ATARI compatibles MS/DOS". Ceci permettra à notre Secrétaire (qui dispose d'un lecteur 3 pouces 1/2 720 ko):

- de reformater votre source selon les critères de la revue pour qu'il sorte comme les autres sur l'imprimante laser...

- de mettre votre source sur le serveur à la disposition des autres adhérents.

Ainsi les sources FORTH, Turbo comme VF-83, seront disponibles sous la même forme (ascii) et utilisables par tous. J'ajoute que:

- un même texte occupe en blocs entre 1 fois et demie et 2 fois plus de place qu'en ascii (une ligne de blancs = 64 caractères au lieu de 2: CR LF ... idem pour les fins de lignes inutilisées) d'où un temps de connection plus court pour récupérer un texte ascii !

- les mots FTH>BLK et BLK>FTH ont été écrits de façon que la double conversion BLK>FTH puis FTH>BLK appliquée à un texte VF-83 (en blocs) restitue un source rigoureusement identique à l'original. Cette double conversion ne présente donc pas d'inconvénient pour nous Ataristes...

■

FORTH

CONVERSIONS ASCII -> BLOCS, et BLOCS -> ASCII

Volksforth-83
Daniel FLIPO

Système: VolksFORTH 83-Standard ATARI
Adaptabilité: néant

Le mot FTH>BLK convertit un texte source TURBO-FORTH (texte ascii) en blocs FORTH de 1024 octets (sans CR ni LF), et inscrit le fichier résultant sur disquette.

Le mot BLK>FTH réalise l'opération inverse: conversion d'un texte FORTH standard (blocs de 1024 octets) en texte ASCII.

Les noms de fichiers peuvent contenir un chemin d'accès. Exemple:

```
source      : A:\DEMOGEM\DEMOGEM.SCR
destination : B:\JEDI\DEMOGEM.FTH
```

Les deux mots vérifient que le nom du nouveau fichier à écrire n'est pas celui d'un fichier déjà existant. Un autre nom est éventuellement demandé pour éviter l'écrasement !

Ndlr: le listing présenté ci-dessous a été retouché par rapport à l'original, afin que chaque ligne ne déborde pas sur la suivante après 58 caractères. Cette retouche ne porte que sur les commentaires, d'où l'aspect "petit nègre" que l'on nous pardonnera.

LISTING: CONVBLK.FTH

```
Ecran 1:
\ Ecran de chargement                          DF 06 08 88

Onlyforth

\needs code          2 loadfrom assemble.scr  cr
\needs compare       include strings.scr
\needs malloc        include allocate.scr
\needs >absaddr      : >absaddr   0 forthstart d+ ;

decimal   2 9 thru

Ecran 2:
Code llfill   ( laddr dlen 8b -- )              \ DF 06 08 88
.w SP )+ D0 move   .l SP )+ D1 move   .l SP )+ A0 move
BEGIN .l  D1 tst  0<> WHILE .b D0 A0 )+ move  .l 1 D1 subq
REPEAT
   Next end-code

 2variable lbuff \ cont. laddr tampon cible(texte formaté)
 2variable sh#   \ offset caractère actuel dans ce tampon
 Create file1  &25 allot  \ contient nom du fichier source
 Create file2  &25 allot  \ contient nom du fichier image.

: +2! ( u addr -- ) \ ajoute u au contenu de var. double
  >r  0  r@ 2@ d+  r> 2! ;
 Onlyforth Dos also forth definitions
: ?file ( -- flag ) \ ?file retourne vrai  si fichier dont
\ nom est à l'adr. file$ existe déjà.
  dta setdta   file2   $10  search0  0=  ;
```

```
Ecran 3:
\ Ecriture d'un fichier sur disquette          DF 06 08 88
Code (writefile ( laddr dlen handle -- d) \ laddr+dlen 32b.
   .w SP )+ D0 move     .l SP )+ D2 move     SP )+ A0 move
      A0 A7 -) move     D2 A7 -) move     .w D0 A7 -) move
   $40 # A7 -) move     1 trap   $0C # A7 adda
   .l D0 SP -) move     Next  end-code

: f_handle   ( -- n ) \ retourne handle du fichier courant
  isfile @ filehandle @  ;
: f_start  ( -- )   \ remet pointeur à début de fichier
  0. f_handle 0 lseek 2drop ;  \ indisp. avant lect/écr.!

: writefile ( laddr dlen -- )  \ écriture fichier courant
  open  f_start   f_handle   (writefile  2dup 0. d<
  IF drop negate ." Erreur d'écriture N°-" . lbuff 2@ mfree
  ELSE 2drop THEN ;

Ecran 4:
\ Création d'un nouveau fichier sur disquette  DF 06 08 88

  Onlyforth

: create_new ( -- )   \ crée un fichier image
  cr ." Nom du fichier à créer : "
  BEGIN file2 dup  &25 erase  &24 expect
       ?file          \ refuser nom fichier existant
  WHILE cr ." Conflit de nom ! Nouveau nom : "
  REPEAT
span @ #tib ! file2 >tib ! >in off \ flot entrée sur file2
  makefile  ;               \ créer nouveau fichier

Ecran 5:
\ Blk_format recopie texte ascii écrit dans un  DF 06 08 88
\ buffer FORTH  en supprimant CR et LF pour faire un bloc
\ Résultat à partir de l'adresse contenue dans lbuff
: blk_format ( -- flag ) \ flag vrai si EOF trouvé EN DEBUT
  capacity 2- 3 * b/blk um*  sh# 2@  d<  \ DE LIGNE
  IF lbuff 2@ mfree  abort" Tampon cible insuffisant" THEN
  0   b/buf  dup b/blk -    caps on
DO drop prev @ I + c@ dup $0D = over $0A = or \ CR ou LF ?
     IF drop sh# 2@ c/l um/mod swap
        IF 1+ c/l um* sh# 2! ELSE drop THEN
        prev @ I + 1+ c@ $0A = IF 0 2 ELSE 0 1 THEN
    ELSE sh# 2@ c/l um/mod drop 0=   \ fin si EOF trouvé
          IF " EOF" count prev @ I + compare 0= \ en début
               IF drop 1 leave THEN   THEN   \ de ligne
   lbuff 2@ sh# 2@ d+ lc!  1 sh# +2!  0 1 \ stocke caract.
     THEN   +LOOP  ;

Ecran 6:
\ Transfert du nouveau fichier sur disque  DF 06 08 88

: save_file     ( -- )
  $6EE3 off \ suppression des signatures (cf. ci-dessous)
create_new  sh# 2@ b/blk um/mod swap IF 1+ THEN  dup  more
  b/blk um*  lbuff 2@   2swap   writefile  ;

\\ Signature des écrans (ID) se trouve à adresse id(Editeur
écran 11), mais l'éditeur compilé sans en-tête cette
adresse plus accessible par id. L'adresse $6EE3 est celle
de id dans 4TH.PRG STANDARD ! Si vous l'avez recompilé avec
un autre STARTUP.SCR vérifiez cette adresse ou gare aux
ADRESS  ERROR  !!!!

Ecran 7:
\ Conversion ASCII -> BLOCS          DF 06 08 88

: FTH>BLK       ( -- )    flush
>tib push  >in push  #tib push  \ sauver pointeurs entrée
  cr ." Nom du fichier à convertir : "
  file1  dup &25 erase  &24 expect
  span @ #tib !  file1 >tib ! >in off \ détour. flot entrée
  use              \ rendre ce fichier courant
capacity 3 * b/blk um* 2dup  malloc  2dup lbuff 2! \ créer
  2swap bl llfill     0. sh# 2! \ tampon de conversion
  capacity  0                  \ conversion
  DO I block drop  blk_format  IF leave THEN  LOOP
  save_file                   \ transfert sur disque
  lbuff 2@ mfree  flush    ; \ libère tampon fermer fichiers

Ecran 8:
\ Conversion BLOCS -> ASCII          DF 06 08 88

: +crlf   ( -- ) \ ajoute CR LF à la position du pointeur
  $0D0A  lbuff 2@  sh# 2@  d+  l!
  2 sh# +2!  ;

: line_format ( adr -- adr1 ) \ adr  = début ligne
  c/l -trailing     \ adr1 = début ligne suivante
 over >absaddr lbuff 2@ sh# 2@ d+ 4 pick lcmove \ recopier
  dup 0= IF 1 sh# +2! THEN    \ ajouter blanc si ligne vide
  sh# +2!   +crlf    c/l +  ;

Ecran 9:
\ Conversion BLOCS -> ASCII          DF 06 08 88

: BLK>FTH     ( -- )    flush
  >tib push  >in push  #tib push \ sauver point. d'entrée
  cr ." Nom du fichier à convertir : "
  file1  dup &25 erase  &24 expect
span @ #tib ! file1 >tib ! >in off \ détour.flot d'entrée
  use                    \ rendre ce fichier courant
  capacity b/blk um*  2dup  malloc  2dup lbuff 2! \ créer
  2swap bl llfill      0. sh# 2! \ tampon de conversion
  capacity 0 DO I block            \ conversion
      l/s 0 DO line_format LOOP  drop  +crlf  LOOP
" EOF " 1+ >absaddr lbuff 2@ sh# 2@ d+ 4 lcmove \ ajouter
  4 sh# +2!  +crlf          \ EOF CR LF
  save_file                 \ sauvegarder le fichier
  lbuff 2@ mfree  flush ; \ libérer tampon+fermer fichiers

EOF
```

VIRGULE FLOTTANTE SUR ATARI ST

par Daniel FLIPO

Système: VolksFORTH 83-Standard pour ATARI

Après avoir téléchargé le pack virgule flottante FPACK.FTH du Turbo-Forth et converti ce fichier ascii en blocs grâce au mot FTH>BLK (voir téléchargement sur Atari ST), il reste à adapter le fichier FPACK.BLK ainsi obtenu pour le rendre réellement utilisable par VF-83.

Le plus simple est de créer un autre fichier FPACK.SCR qui contiendra le source adapté à VF-83 : l'écran 0 pourra contenir des commentaires, l'écran 1 assurera le chargement, les mots S-> et <-S seront réécrits en assembleur 68000 à l'écran 2, et on transfèrera à partir de l'écran 3 la partie utile du fichier FPACK.BLK (celle qui commence par le mot F!).

Ce transfert se fait de manière très commode grâce aux commandes de l'éditeur: on tapera USE FPACK.SCR 3 L <return> pour passer sous éditeur, puis CTRL M pour "marquer" cet écran, CTRL S pour sortir de l'éditeur, enfin USE FPACK.BLK 4 L <return> pour charger l'écran 4 de FPACK.BLK.

Dès lors la commande CTRL A permet de basculer d'un fichier à l'autre: on prélève des lignes par CRTL "flèche en bas" ou SHIFT "flèche en haut" dans FPACK.BLK, on passe à l'écran courant dans FPACK.SCR par CTRL A et on y dépose les lignes prélevées par SHIFT "flèche en bas", on revient dans FPACK.BLK par CTRL A et ainsi de suite...

On peut aussi transférer des blocs entiers grâce aux mots COPY et CONVEY Exemple: FPACK.SCR (destination) 4 5 from FPACK.BLK 3 convey recopie les écrans 4 et 5 de FPACK.BLK à partir de l'écran 3 de FPACK.SCR

Voici le listing des écrans 1 et 2 :

```
\ Ecran 1 : chargement du programme

 ONLYFORTH
 Vocabulary FLOAT
 \needs code        2 loadfrom assemble.scr
 ONLYFORTH  FLOAT  ALSO  DEFINITIONS
```

```
2 capacity 1- thru

\ Ecran 2 : Routines de décalage 32 bits

 Code (-S  ( ud bit-entrée-D -- ud1 bit-sortie-G )
.w  SP )+ D1 move   .l  SP )+ D0 move
.b  1  #  D1 lsr    .l  1  #  D0 roxl
                      \ mise à jour X et rot.
.b  1  #  D1 roxl     \ récupérer bit CARRY
.l  D0 SP -) move   .w  D1 SP -) move   Next  end-code

 Code S-)  ( ud bit-entrée-G -- ud1 bit-sortie-D )
.w  SP )+ D1 move   .l  SP )+ D0 move
.b  1  #  D1 lsr    .l  1  #  D0 roxr
                      \ mise à jour X et rot.
.b  1  #  D1 roxl     \ récupérer bit CARRY
.l  D0 SP -) move   .w  D1 SP -) move   Next  end-code
```

Quelques mots devront être réécrits, par exemple FDUP car le mot 3DUP n'existe pas en VF-83 :

```
    : FDUP    2 pick  2 pick  2 pick ;
```

ou mieux en assembleur 68000 :

```
    Code FDUP  ( n1 n2 n3 --- n1 n2 n3 n1 n2 n3 )
        .l  2 SP D) SP -) move   \ recopier n1 n2
        .w  4 SP D) SP -) move   \ recopier n3
         Next end-code
```

Il manque également :

```
    : between ( n [low up[ -- flag )
        rot under ) -rot ) not and ;
    : ?enough ( u   -- )
        depth ) abort" Pas assez d'arguments" ;
    : 1+!      ( adr -- )  1 swap +! ;
    : 1-!      ( adr -- ) -1 swap +! ;
```

ou mieux en assembleur :

```
    Code 1+!  SP )+ D6 move   D6 reg) A0 lea   1 A0 ) addq
    Next  end-code
    Code 1-!  SP )+ D6 move   D6 reg) A0 lea   1 A0 ) subq
    Next  end-code
```

Certains mots portent des noms différents en VolksFORTH-83: S)D et MU/MOD existent en VF-83 sous les noms de EXTEND et UD/MOD respectivement. On écrira:

```
    ' extend Alias s)d
    ' ud/mod Alias mu/mod
```

ainsi les CFA de extend et ud/mod seront directement placés à chaque occurence de s)d et mu/mod, ce qui évite le renvoi inutile que provoqueraient les définitions équivalentes

```
    : s)d    extend ;
    : mu/mod ud/mod ;
```

Enfin la définition du mot VAL devra être réécrite car VF-83 ne dispose pas du mot (NUMBER?) mais de NUMBER qui exécute un peu différemment la conversion d'une chaine de caractères en nombre sur 32 bits.

```
    : val ( adr -- f )
      base push decimal
      ascii + skip
      2dup ascii E scan )r )r
      drop r@ over - over 1- c! 1- number \ ligne modifiée
      dpl @ 1- = IF drop THEN E
      r) r) ?dup IF ascii E skip  ascii + skip
                    over 1- c! 1- number drop
                                       \ ligne modifiée
              ELSE drop 0
              THEN       float ;
```

On notera le rôle du mot push pour la sauvegarde de la base...

Moyennant ces adaptations le module FPACK fonctionne très bien sur ATARI ST. Les puristes pourront gagner en vitesse d'exécution en réécrivant les mots FDROP FSWAP FOVER FROT en assembleur (facile en s'inspirant des définitions de 2DROP 2SWAP... du fichier FORTH-83.SCR.),et pour les plus courageux aussi D*, F*, F/, F+ et F- (exercice nettement plus coton !!). ■

CREATION D'UN ACCESSOIRE

par Daniel FLIPO

Système: VolksFORTH 83-Standard ATARI

Les accessoires sont des programmes sous GEM de suffixe .ACC accessibles par la barre de menu même lorsqu'un autre programme sous GEM est en cours.

Les programmes .ACC présents sur le répertoire principal du disque A sont chargés à la mise sous tension ou au reset de l'ATARI (6 au maximum)et restent résidents jusqu'au prochain reset. Ils attendent un signal de lancement (clic sur leur nom dans le menu) pour s'exécuter.

L'installation se fait par les mots ACC_INIT et BOOTCODE, l'accessoire proprement est constitué par le mot RUNACC : c'est une boucle sans fin BEGIN run wait REPEAT (sans WHILE !) où:

- RUN est le mot FORTH à exécuter
- WAIT attend la prochaine sollicitation de l'accessoire.

Le mot SAVEACC est à la création d'accessoires ce que SAVESYSTEM est à la création de programmes .PRG à deux différences près:

- le chargement du fichier de ressources (.RSC) ne doit pas être fait dans de mot FORTH à exécuter (il se rechargerait à chaque appel de l'accessoire et aurait tôt fait de bloquer toute la mémoire !) mais une fois pour toute à l'installation. C'est le mot RSRC$ qui s'en charge.

- il est nécessaire d'intaller le nom donné à l'accessoire dans le menu par le mot TITLE$.

Ndlr: le listing présenté ci-dessous a été retouché par rapport à l'original, afin que chaque ligne ne déborde pas sur la suivante après 58 caractères. Cette retouche ne porte que sur les commentaires, d'où l'aspect "petit nègre" que l'on nous pardonnera.

LISTING: SAVEACC.FTH

```
Ecran 1:
\\            ***  SAVE_ACC.SCR  ***           DF 12 02 88

Ce fichier met en place la définition du mot  saveacc  qui
permet  de  sauvegarder  un   programme  sous  forme
d'accessoire.
      Utilisation : compiler les mots
: rsrc$  ( --- addr ) 0" Nom.rsc" ; \ Nom fich. ressources
\ s'il y  en  a  1,  sinon  supprim.  rsrc$  dans acc_init
(écr.4)
: title$ ( --- addr) 0"  Nom_accessoire" ; \ Nom qui devra
\ figurer dans le menu (laisser 2 blancs devant ce nom).
puis, si le mot FORTH à exécuter est ESSAI , faire :

     ' ESSAI alias  run     (return)
     include SAVE_ACC.SCR  (return)
     saveacc NOM.ACC       (return)

Grand merci à Dietrich WEINECK et Bernd PENNEMANN,
coauteurs VolksFORTH-83, qui m'ont aimablement communiqué
la base de ce programme de lancement d'un accessoire !
```

FICHIER: SAVE_ACC.FTH

```
Ecran 2:
\ Ecran de chargement                              DF 13 02 88

  Onlyforth
\needs Code             2 loadfrom assemble.scr
\needs gem              include gem\basics.scr

  Onlyforth Gem also
\needs menu_register    2 load
\needs evnt_multi       : evnt_multi    &25 &16 7 1 AES ;
\needs message          create message  $10 allot
\needs :mu_mesag        $10  >label  :mu_mesag
\needs :acc_open        &40  >label  :acc_open

 3 5 thru

Ecran 3:
\ Fonction menu_register                           DF 09 02 88

| : ?error ( id --- )
    0< abort" Plus de 6 accessoires" ;

| : menu_register ( apid laddr --- id )
    addrin 2!  intin !  &35 1 1 1 AES  dup  ?error ;

Ecran 4:
\ Attente du signal d'ouverture                    DF 09 02 88
| Variable acc_id                    \ N° de l'accessoire
| Create acc_events
   -1         ,            \ tous évènements acceptés
   0 , 0 , 0 ,                    \ pas d'évènement souris
   here $14 allot $14 erase  \ rectangles non spécifiés
   0 , 0 ,                  \ 0 milliseconde délai timer

| : acc_prepare  ( --- )
  acc_events intin $20 cmove  message >absaddr addrin 2! ;

| : wait  ( --- )    clsvwk
 BEGIN
   BEGIN  pause acc_prepare  evnt_multi :mu_mesag and UNTIL
     message @ :acc_open =  message 8 + @ acc_id @ =  and
 UNTIL  opnvwk ;

Ecran 5:
\ Initialisation de l'accessoire                   DF 13 02 88

| : ap_id  global 4 + @ ; \ identificateur de l'accessoire

| : runacc   BEGIN  run  wait  REPEAT ;  \ boucle sans fin

| : acc_init ( --- )          \ initialisations
  grinit
  rsrc$  rsrc_load      \ charger le fichier ressources,
\ voir la déf. de RSRC$ à l'écran 0.
ap_id  title$  >absaddr menu_register acc_id ! \ installer
\ le nom dans le menu (cf déf. de TITLE$ à l'écran 0)
  wait   ['] noop Is 'cold   runacc  ;

' acc_init is  'cold

Ecranb 6:
\ En-tète de l'accessoire                          DF 12 02 88

: saveacc  ( --- )         \ sauvegarde de l'accessoire
limit here - 0  $400. d+ $0A 2! \ taille du segment bss ( +
  savesystem  ;                  \ 1 Ko pour la pile )

Label bootcode
  .l bootcode $100 - pcrel) A5 lea  \ Début de page de base
  .l $18 A5 D) D0 move               \ BSS-Start
     $1C A5 D) D0 add          \ ajouter la longueur du BSS
     $4. #   D0 sub            \ laisser la place pour 1mot
     D0 A7 move                \ mise à jour pointeur pile
  .l bootcode $1C - pcrel) FP lea   \ FP sur Forthstart
     ' cold >body r#) jmp

 bootcode  $1C   here bootcode -  cmove

EOF
```

FORTH

FORMATAGE SUR ATARI ST

Daniel FLIPO

Système: VolksFORTH 83-Standard ATARI

QFORMAT permet de formater une disquette simple ou double face, avec 9 ou 10 secteurs par piste et 80 pistes par face, au standard ATARI ou MS/DOS 3.2, et éventuellement d'accélérer la lecture et l'écriture des fichiers inscrits ultérieurement.

Contrairement à d'autres programmes décrits ailleurs (Livre du lecteur de disquettes p. 48, ou Développer en GFA BASIC) les secteurs défectueux sont répertoriés et signalés sur la FAT afin qu'aucune donnée ne soit ultérieurement enregistrée sur eux.

I) PRINCIPE DU FORMATAGE

A) Structure d'une disquette

Une disquette comporte normalement 80 pistes numérotées de 0 à 79 en simple face ou 160 numérotées de 0 à 159 en double face. En double face les pistes sont disposées alternativement sur les deux faces: pistes paires sur la face zéro, pistes impaires sur la face 1.

Une piste contient en principe 9 secteurs de données (éventuellement 10) et différentes informations système telles que gap etc... Chaque secteur contient 512 octets.

Le formatage a un double but: écrire les informations système nécessaires au repérage des pistes et des secteurs, et tester la qualité des secteurs de données. Les 18 premiers secteurs (pistes 0 et 1) sont réservés:

- -Secteur 1 : Boot-sector
- -Secteurs 2 à 6 : 1ère copie de la FAT (File allocation table)
- -Secteurs 7 à 11 : 2ème copie de la FAT (par sécurité)
- -Secteurs 12 à 18 : Répertoire principal.

B) Fonctions du XBIOS nécessaires au formatage.

- -FLOPWR (N°9) : écriture de secteurs d'une même piste.
- -FLOPFMT (N°10) : formatage d'une piste
- -PROTOBOOT (N°18): prototype du Boot-sector.

Les appels au XBIOS sont faits aux écrans 4 à 6. La fonction FLOPFMT demande différents paramètres en entrée, dont le facteur d'entrelacement (interleave en anglais) qui indique dans quel ordre les secteurs doivent être lus. Normalement sa valeur est 1, mais on peut accélérer les opérations de lecture et d'écriture sur le disque en lui donnant la valeur 11.

Avec le facteur 1, la tête de lecture trouve en début de piste les informations dont elle a besoin et en particulier le numéro de la piste où elle se trouve. Si c'est la piste cherchée elle doit reprendre la lecture au début, et donc faire un tour pour rien ! Avec le facteur 11 le numéro de piste est écrit sur le 11ème secteur (dégénéré) et la lecture de la piste suivante peut commencer immédiatement...

Ces explications théoriques sont les seules dont je dispose, mais elles ne me satisfont pas pleinement ! Néammoins, on constate expérimentalement qu'un fichier long enregistré d'un seul tenant sur des pistes consécutives est écrit et lu jusqu'à deux fois plus vite. Si le même fichier est morcelé (cas d'un enregistrement sur une disquette où des fichiers éffacés ont laissé des "trous") le gain peut être négligeable ... J'ai bien sûr essayé d'autres valeurs pour le facteur d'entrelacement (2, 5, 10 ..) c'est 11 qui donne les meilleurs résultats.

C) Formatage.

Deux tampons sont nécessaires: un qui commence à l'adresse (32 bits)pointée par la variable form_buff et qui doit avoir une capacité de 8 ko minimum, et un qui commence à l'adresse (32 bits) pointée par fat_buff de capacité min. $A00=5 * 512 octets.

Dans le premier la fonction FLOPWR prépare le formatage d'une piste complète (données + gaps) et dans le second nous construirons un prototype de la FAT (5 secteurs de 512 octets). J'ai porté la capacité du premier tampon à 9 ko pour pouvoir y stocker la totalité des 18 premiers secteurs.

Le formatage d'une face est effectué par le mot FORMAT_SIDE (écran 14). La fonction FMT_TRACK est appelée pour chaque piste et retourne sur la pile un 16-bits qui vaut 0 si tout s'est bien passé, ou un nombre compris entre -1 et -17 suivant le type d'erreur survenu. Ce nombre est récupéré par le mot ERROR_DEAL pour traitement: soit affichage d'une boite d'alerte (volet de protection fermé, absence de disque etc.) soit repérage d'un secteur défectueux si le nombre retourné est -16 ($FFF0).

Le mot FMT_TRACK remplit les secteurs de données avec des octets $E5 = %11100101. Les 0 et 1 sont répartis de manière suffisamment irrégulière pour que cette valeur constitue un bon test de la qualité d'un secteur.

La fonction FLOPWR relit le secteur et si un bit n'est pas enregistré correctement elle retourne la valeur -16 et écrit au début du tampon fmt_buff les numéros des secteurs défectueux de la piste. Exemple: Si les secteurs 1, 2 et 6 sont défectueux le début du tampon contiendra les octets:

00 01 00 02 00 06 00 00 4N 4N 4N 4N ....

Le mot BAD_CLUSTER enregistre ces informations sur le tampon fat_buff d'où elles seront reportées sur la FAT afin que ces secteurs ne soient pas utilisés par la suite.

D) Structure de la FAT.

Chaque groupe de 2 secteurs consécutifs constitue une entité nommée CLUSTER qui contient donc 1024octets. La FAT contient les informations utiles sur tous les clusters du disque, codées SUR 12 BITS soit 3 chiffres en hexadécimal!!!

$000 signifie cluster disponible,
$FFF signifie dernier cluster d'un fichier,
$FF7 signifie cluster défectueux,

et tout nombre compris entre $002 et $FF6 indique le numéro du cluster où se trouve la suite du fichier. Penser qu'un fichier n'est pas nécessairement enregistré sur des clusters consécutifs: au contraire on bouche les "trous" laissés disponibles par les fichiers éffacés...

Les 3 premiers octets de la FAT doivent contenir au départ les valeurs $F7 $FF $FF. Le reste de la FAT est découpé en groupes de 3octets, chaque groupe contenant les informations sur 2 clusters consécutifs (3 octets = 24 bits = 2 fois 12 bits !). Le décodage d'un groupe de 3 octets se fait de la manière suivante; exemple de début d'une FAT:

F7 FF FF puis
03 40 00 : 05 60 00 : 09 70 FF : F7 AF 00 : 0B F0 FF ..

Soit le 1er groupe: On prend le 4ème chiffre hexa (0), on le fait passer devant le 1er, et on fait passer le 3ème (4) derrière le dernier et 03 40 00 devient 003 004 ! Le 1er cluster UTILISABLE est numéroté 002 et non 001 (?) et c'est en fait le 19ème cluster puisque les 18 premiers sont occupés par le boot-sector, les FATs et le répertoire.

Dans notre exemple le 12 bits représentant ce cluster contient la valeur 003, ce qui signifie que la suite du fichier commencé au cluster 2 se trouve au cluster 3 (le suivant). Le 12 bits représentant le cluster 3 contient 004, la suite du fichier est donc dans le cluster 4 et ainsi de suite...

Décodons les groupes suivants:

2ème groupe 05 60 00 donne 005 006 pas de problème,
3ème groupe 09 70 FF donne 009 FF7 ah tiens!
4ème groupe F7 AF 00 donne FF7 00A
5ème groupe 0B F0 FF donne 00B FFF

Le fichier est donc enregistré sur les clusters 2, 3, 4, 5, 6, 9, 10, 11 et c'est tout puisque le cluster 11 est désigné par FFF(fin de fichier). Les clusters 7 et 8 sont défectueux et le système n'y a donc rien enregistré...

Il faut savoir enfin trouver quel est le premier cluster occupé par un fichier: c'est le répertoire principal qui le donne. Il contient les noms de tous les fichiers et dossiers présents sur le disque et divers informations les concernant (date et heure de mise à jour, taille etc.) dont le numéro du 1er cluster occupé par le fichier (offset $1A, numéro sur deux octets cette fois !). On aurait trouvé $0002 dans l'exemple ci-dessus.

On dispose ainsi, grâce à la FAT et au répertoire principal d'une table des matière complète de la disquette.

Remarques (cyniques) :

1) Le découpage en clusters et le codage sur 12 bits sont malcommodes mais justifiés (?) par le gain de place obtenu: une disquette comporte au maximum 2 faces de 80 pistes et 10 secteurs par pistes soit 800 clusters. Leur codage sur 8 bits est impossible mais sur 12 ça marche! Ainsi la FAT tient sur 1200 octets soit moins de 3 secteurs. Si on avait codé les clusters sur 16 bits, elle aurait occupé au maximum 1600 octets soit moins de 4 secteurs, et de toute manière Atari réserve 5 secteurs pour chaque FAT, pour des applications futures???

2) Le formateur en ROM accessible par le bureau GEM inscrit des $FFF pour les clusters défectueux et non des $FF7 comme l'affirment les documentations, allez savoir pourquoi ...

Si une erreur -16 se produit,le mot BAD_CLUSTER lit le début du tampon de formatage jusqu'à rencontrer la valeur 0 et inscrit $FF7 (sous la forme F7 F_ ou 7_ FF selon le cas!) à l'endroit voulu: le numéro du groupe de 3 octets concerné (1 pour le 1er cluster utilisable) est donné par la formule: ((N°piste * Nbre-de secteurs/pistes + N°secteur-déf. -19)/ 4 + 1). / désigne la division euclidienne (quotient entier). Si un des 18 premiers secteurs est défectueux la disquette est inutilisable, une boite d'alerte le signale.

E) Structure du boot-sector.

La fonction XBIOS PROTOBOOT crée une image du boot-sector standard (1 ou 2 faces mais seulement 9 secteurs par piste). Si on lui fournit un nombre de 32 bits supérieur à $FFFFFF elle génère automatiquement un nombre aléatoire de 24 bits qui sert de N° de série et permet de détecter les changements de disquette.

Lorsqu'on formate en 10 secteurs par piste, il y a lieu de modifier les nombres NSECTS (nombre total de secteurs du disque, offset $13) et le nombre SPT (nombre de secteurs par piste, offset $18).

La structure complète de boot-sector est décrite dans les Clés pour ATARI ST p. 182, et dans le livre du lecteur de disquettes p. 44.

\\ FORMATAGES DIVERS DF 06 08 88

QFORMAT propose diverses options de formatage: simple/double face, 9 ou 10 secteurs par pistes, format Atari ou MS/DOS (les disquettes ainsi formatées sont utilisables indifféremment sur Atari ou sur les compatibles IBM), et choix du facteur d'entre-lacement (INTERLEAVE). Le facteur normal est 1 mais le choix d'une autre valeur (11 ou 6 par exemple) permet sur

certains lecteurs d'accélérer notablement les opérations de lecture et d'écriture sur disquette. Le résultat est variable selon les modèles, il convient de faire des essais: lire et écrire un fichier assez long (> 100 ko) sur une disquette vierge afin que le fichier soit sur des pistes consécutives de la disquette.

L'opération de formatage prend environ 50 secondes par face, comme le formatage standard (entrelacement=1) du desktop.

Ecran 1:

```
  Onlyforth      \ Ecran de chargement             DF 07 08 88
\needs >label        2 loadfrom assemble.scr
\needs case          include case_of.scr
\needs lfill         include long.scr
\needs gem           include gem\basics.scr
  Onlyforth gem also definitions
\needs objc_draw     &13 loadfrom gem\aes.scr
\needs form_do       &17 loadfrom gem\aes.scr
\needs rsrc_load"    &34 loadfrom gem\aes.scr
\needs tree!         include gem\supergem.scr
\needs mofaddr       2variable mofaddr    0. mofaddr 2!
: graf_mouse ( n -- )
 intin ! mofaddr 2@ addrin 2! &78 1 1 1 AES
                     0= abort" Graphic_error " ;
 cr  2  &15  thru
\      &16  load      ( Pour installation en .PRG )
\      &17  load      ( Pour installation en .ACC )
```

Ecran 2:

```
\ Transcription du fichier QFORMAT.H            DF 06 08 88
 0   >label :opt        5   >label :A        4   >label :B
 7   >label :df         8   >label :sf       &12 >label :std
 9   >label :ok         &10 >label :no       &13 >label :10s
 &11 >label :sect       &16 >label :ata      &17 >label :ibm
 &18 >label :ilv        &20 >label :il1      &21 >label :fact
 &22 >label :name

 1   >label :wait       2   >label :rect1    3 >label :hach1
                        4   >label :rect2    5 >label :hach2
 2   >label :free       5   >label :f1       6 >label :f2
                        4   >label :disp

\ Boites d'alerte
 0   >label :hs      1   >label :prot    2    >label :nodk
 3   >label :div     4   >label :lect
```

Ecran 3:

```
  variable ibm            variable il      \      DF 06 08 88
  variable track          variable disk       variable #sect
  variable dble           variable side       variable :h
  variable bs1            variable bs2        variable bc
 2variable fat_buff      2variable form_buff

  Create empty$    3 c, bl c, bl c, bl c,

: memalloc ( --- )      \ réservation mémoire
  $A00. malloc fat_buff 2!   $2400. malloc  form_buff 2! ;
: settings ( --- )  memalloc  rsrc_load" qformat.rsc"  ;
: clexit   ( --- )       \ pour sortir proprement ...
 fat_buff 2@ mfree  form_buff 2@ mfree  rsrc_free  grexit ;

: >f ( objNr --- flag) \ fl=1 si radiobutton select 0 sinon
  state_gaddr l@ ;
```

Ecran 4

```
\ Ecriture d'une piste (Fonction XBIOS 9)        DF 06 02 88
Code (write_track ( laddr disk 1st track side #sect --- n )
   .w SP )+ A7 -) move       \ Nbre de secteurs à écrire
      SP )+ A7 -) move       \ N° face (0 ou 1)
      SP )+ A7 -) move       \ N° de piste (0 à79)à écrire
      SP )+ A7 -) move       \ N° 1er secteur à écrire
      SP )+ A7 -) move       \ N° du disque (A=0,B=1)
   .l 0. #  A7 -) move \ Valeur bidon (utilisation future)
   .l SP )+ A7 -) move       \ Adresse (32bits) du tampon
   .w 9  #  A7 -) move       \ Appel fonction  FLOPWR  (N°9)
      $0E   trap             \ du XBIOS
      $14 # A7 adda          \ Restauration de la pile A7
      D0 SP -) move      \ N° d'erreur(16bit<0) ou 0 si OK
   Next end-code
: write_track ( laddr disk track side --- n ) \ Ecr. piste
  1  -rot  #sect @  (write_track   ;            \ complète
```

Ecran 5:

```
\ Format. d'1 piste (laddr -> tampon 8 KO min) DF 06 08 88
Code fmt track ( laddr disk track side --- n )
  .w  $E5E5 # A7 -)  move     \ octet de remplissage
  .l  $87654321. # A7 -) move   \ Nombre "magique"
  .w  il R#)  A7 -)  move     \ Facteur d'entrelacement
      SP )+   A7 -)  move \ N° face (0 ou 1) pris sur pile
      SP )+   A7 -)  move \ N° piste(0 à 79) pris sur pile
    #sect R#) A7 -)  move \ Nombre de secteurs par piste
      SP )+   A7 -)  move \ N° disk(A=0,B=1) pris sur pile
  .l  0. #    A7 -)  move \ val. bidon mais indispensable!
      SP )+   A7 -)  move \ laddr du tampon prise sur pile
  .w  $A #    A7 -)  move \ appel de la fonction FLOPFMT
      $E      trap            \ (N°10) du XBIOS
      $1A  #  A7     adda     \ restauration de la pile A7
      D0      SP -)  move     \ N° d'erreur (16b.<0) ou 0
next  end-code
```

Ecran 6:

```
\ Construction "BOOT-SECTOR" standard (9 sect.)DF 06 08 88
\ A l'appel de PROTOBOOT le tampon commencant à laddr doit
\ contenir un sect. "préformaté" (fabriqué par FMT_TRACK).

Code protoboot ( laddr n ---) \ crée 1 image boot-sector
.w 0 #  A7 -) move   \ pas de prg. exécut. sur boot-sector
     SP )+ A7 -) move \ simple (n=2) ou double (n=3) face
  .l $1000000. # A7 -) move \ N° série aléat. sur 24 bits
     SP )+ A7 -) move \ laddr d'un tampon de 512 oct. min.
  .w $12 # A7 -) move    \ appel de la fonction PROTOBOOT
     $E  trap            \ du XBIOS
     $E  # A7 adda       \ restauration de la pile
next end-code


\\ Pour 1 format. non-standard (10 sect/piste) le protot.
de boot-sector ainsi créé doit être modifié (cf écran 13)
```

OPTIONS DE FORMATAGE

DISQUE A — DISQUE B

SIMPLE FACE — DOUBLE FACE

STANDARD — 10 SECTEURS

ATARI — MS/DOS

Entrelacement :

STANDARD — Facteur : 06

Nom du disque : DISQUE_1___

FORMATER — ANNULER

fig 1.

Ecran 7:

```
\ Gestion des boites de dialogue et d'erreur   DF 20 01 88
\ Btes dialogue: box# = N°(>label) de la boite, field#=
\ N° du 1er champ texte éditable ou -1 s'il n'y en a pas .

: dialog   ( field# box#  --- bouton )
  tree!  curoff  hide_c  show_object
  form_do dup deselect hide_object curon   ;

: show_info ( --- ) :wait tree! curoff hide_c ( show_objc)
```

```
init_object big 4@ scr>mem1 1 little 4@ big 4@ form_dial
0 1 big 4@ objc_draw show_c ; \ =show_objc mais profond. 1
: hide_info ( --- )   :wait tree! hide_object curon ;

: prep_info ( --- x y ) \ retourne coordon. du coin haut
  :wait tree!  side @   \ gauche du rectangle à hachurer
  IF :hach2 :h ! ELSE :hach1 :h ! THEN :h @ objc_offset ;
```

Ecran 8:

```
\ Gestion de la boite OPTIONS DE FORMATAGE     DF 06 08 88

: disable ( obj# -- ) \ déselect. et met en grisé un objet
  %1000 swap state_gaddr l!  ;

: enable  ( obj# -- ) \ permet sélection ultér. d'un obj.
  state_gaddr 2dup  l@ %110111 and  -rot  l!  ;

: objc_redraw ( obj# -- ) \ redessine un objet modifié
  3  over objc_offset  3 pick objc_getwh  objc_draw  ;

: store_options ( -- )    \ initialise variables de choix
  :opt tree!  :B >f disk !  :df >f dble !
  :std >f  IF 9 ELSE &10 THEN #sect !
  :il1 >f  IF 1 ELSE :fact getnumber drop THEN il !
  :ata >f  IF ibm off ELSE ibm on THEN  ;
```

Ecran 9:

```
\ Gestion de boite OPTIONS DE FORMATAGE (fin) DF 15 09 88
: options_choice ( -- flag ) \ fl vrai si format. demandé
  :opt tree! curoff hide_c show_object
  BEGIN  :name form_do   CASE
:10s OF :il1 select :fact disable :ilv objc_redraw 0 ENDOF
:std OF :fact enable              :ilv objc_redraw 0 ENDOF
:ibm OF :il1 select :fact disable :ilv objc_redraw
        :name disable             :name objc_redraw
     :std select  :10s  disable :sect objc_redraw 0 ENDOF
:ata OF :fact enable               :ilv  objc_redraw
        :name enable               :name objc_redraw
        :10s  enable             :sect objc_redraw 0 ENDOF
:ok  OF store_options
        :ok deselect  hide_object  curon    -1  -1 ENDOF
:no  OF :no deselect  hide_object  curon     0  -1 ENDOF
    ENDCASE          UNTIL  ;
```

Ecran 10:

```
\ Inscript. des secteurs défectueux sur la FAT DF 06 08 88

: fat_init ( --- )     \ préparation du buffer de FAT
  fat_buff 2@  2dup  $A00 0 lfill
  $F7  ibm @ IF dble @ 1+ + THEN
  $FFFF  2over 1. d+ l!  -rot lc! ;

: 1fat! ( u --- ) \ écriture de F7  F     dans fat_buff
  0  fat_buff 2@  d+  2dup  2dup  lc@ 0= IF 1 bc +! THEN
  $F7  -rot  lc!  1. d+ 2dup lc@  $0F  OR  -rot  lc! ;

: 2fat! ( u --- ) \ écriture de    7  FF dans fat_buff
  1+ 0  fat_buff 2@ d+  2dup  2dup  lc@  $70  OR  -rot  lc!
  1. d+  2dup  lc@ 0= IF 1 bc +! THEN  $FF -rot lc!  ;
```

Ecran 11:

```
\ Repérage des secteurs défectueux              DF 13 02 88
: bad_cluster ( --- :hs 0  ou  1 )
  track @  dup 0=  swap dble @ 1+ =  OR
  IF :hs 0          ELSE  form_buff 2@
  BEGIN  2dup l@
  WHILE 2dup  1. d+ lc@ track @ #sect @ * + &19 - 4 /mod 1+
        3 *  swap 1- 0) IF 2fat!  ELSE 1fat!  THEN
          1  side @  IF bs2 +! ELSE bs1 +! THEN  2. d+
  REPEAT 2drop  1  THEN  ;

\\ Pour  visualiser  les  N°  des  secteurs  défectueux (il
s'inscrivent dans  form_buff), ajouter le mot view_bad dans
error_deal juste avant bad_cluster (écran 10)
: view_bad ( --- )     \ ldump est défini dans tools.scr !
  cr ." Piste : " track @ .
  form_buff 2@  $20  ldump   cr  key drop ;
```

Ecran 12:

```
\ Gestion des erreurs                           DF 06 08 88

: error_deal ( n --- )
  ?dup IF  CASE  $FFF0    OF    bad_cluster  IF  exit THEN
ENDOF
                 $FFF3   OF    :prot
ENDOF
                 $FFFD  OF  :lect
ENDOF
                 $FFFA    OF    :nodk
ENDOF
                 $FFEF  OF  :nodk
ENDOF
                 :div    swap ENDCASE
            hide_info  curoff  alert  abort  THEN ;

\\ Dans le mot vectorisé 'ABORT qui est exécuté dans ABORT
on placera  le CFA  du mot  qui relancera  le formatage au
début:
 QFORMAT par exemple .... (cf écrans 15, 16 et 17). Le mot
QUIT executé ensuite (dans  ABORT) empêchera  le formatage
de
 recommencer au niveau  de l'erreur !
```

Ecran 13:

```
\ Construction  des 2  premières pistes (face 1)  DF 15 09
88
 : create_boot ( --- ) \ créat. "boot-sector" sur form_buff
   form_buff 2@  dble @ 2+  protoboot      #sect @ $A =
   IF form_buff 2@ $13. d+  $A  $2003 dble @ 1+ *  2over l!
   -rot  $5. d+ lc!  THEN \ modifications si 10 sect./piste
   ibm @  IF $55AA  form_buff 2@ $1FE. d+  l!
              3        form_buff 2@ $16.  d+  lc! THEN  ;
 :  create_fat  (  ---  )  \  créat.  des  2  FATs+REP. sur
\ form_buff
   form_buff 2@  fat_buff 2@  2over 2over 2over  ibm @
 IF  $200.  d+  $600  lcmove  $800.  d+ $600 lcmove \ 2FATs
\(3sect
     $E00. d+  $E00 0 lfill           \ répertoire (7sect)
 ELSE $200. d+ $A00 lcmove $C00. d+ $A00 lcmove
   $1600. d+  2dup $E00  0 lfill                \ répertoire
\(7sect)
       :opt tree! :name text_gaddr 2dup lc@ \ nom éventuel du
 IF 2over  $B  lcmove   $0B -rot   $B. d+  lc! \ disque
          ELSE 2drop 2drop THEN        THEN  ;
```

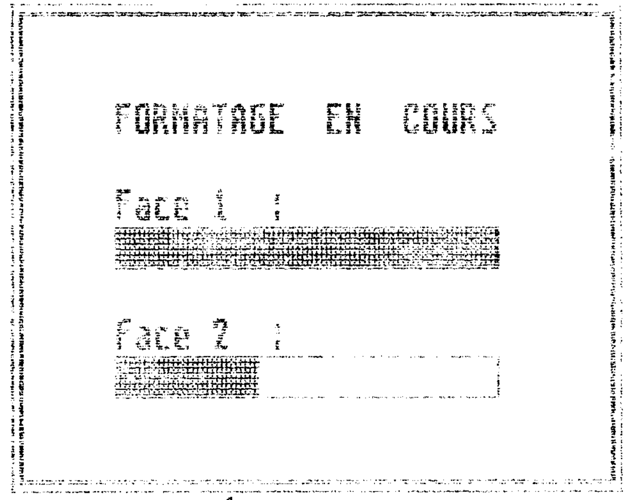


fig. 2

Ecran 14:

```
\ Formatage d'une face                         DF 06 08 88
: format_side ( -- )         \ formatage des pistes 79 à 0
  prep_info  &79 dble @ IF 2* side @ + THEN track ! 80  0
  DO :h @ 1  2over  I 1+ 2*  cheight  objc_draw
form_buff 2@ disk @  &79 I -  side @  fmt_track error_deal
     dble @ 1+ negate  track +!   LOOP  2drop  ;
: write_fat ( --- )    \ écrit. BOOT, FATs, REP. sur disque
  form_buff 2@  disk @  0  0  write_track  error_deal
form_buff 2@  $1200   #sect @ $A =  IF $200 + THEN  0  d+
  disk @  1  1  0  dble @ IF swap THEN   &18  #sect @ -
  (write_track  error_deal ;
\ Initial. du buffer d'écran pour toutes les résolutions
: q_extend ( flag -- )  intin ! &102 0 1 VDI ;
: setMFDB  ( adr  -- )  >r 0 q_extend intout 2@ r@ 4+ 2!
```


JEDI N° 47. Juillet 1988

```
intout @ $10 / r@ 6 + ! 1 q_extend intout
8 + @ r> &12 + ! ;
: rastmemset    ( -- )  memMFDB1 setMFDB ;

Ecran 15:
\ Formatage global du disque                    DF 15 09 88
: Qformat  ( --- )  base  push  decimal   0  graf_mouse
rastmemset  BEGIN  options_choice
  WHILE  side off  track off  bs1 off  bs2 off  bc off
       2 graf_mouse  show_info  fat_init  format_side
       dble @  IF 1 side !  format_side  THEN
       create_boot  create_fat  write_fat  hide_info
       0 graf_mouse  :free tree!  bs1 @ 0 :f1 putnumber
       dble  @  IF  bs2  @  0 :f2 putnumber ELSE empty$ :f2
putstring      THEN   &40 #sect @ *  dble @ 1+  *
       ibm @ IF 9 ELSE $8 THEN -
       bc @  -  $400  um*  :disp  putnumber
       -1 :free dialog  drop
  REPEAT curoff  show_c  ;

   ' qformat Is 'abort
```

Ce disque possède
726016
octets disponibles.

Secteurs défectueux :
Face 1 : 0
Face 2 : 0

OK

fig. 3

```
Ecran 16:
\  Installation en QFORMAT.PRG :                DF 07 08 88
\  Compiler écrans 1 à &16 inclus (supprimer le \ devant
\  &16 load à l'écran 1) à partir de FORTHKER.PRG par :
\  include qformat.scr

 : finish ( --- )  qformat clexit bye ;

 : run     ( --- )  grinit  settings  finish  ;

 ' finish Is 'abort

 ' run Is 'cold

 savesystem QFORMAT.PRG


Ecran 17:
\ Installation en QFORMAT.ACC  :                DF 07 08 88
\ Compiler les écrans 1 à &15 ci-dessus puis l'écran &17
\ (supprimer le \ devant  &17 load  à écran 1) à partir de
\ FORTHKER.PRG par :   include QFORMAT.SCR

: rsrc$  ( --- addr ) \ réservat. mém. et nom "ressource
  memalloc   0" qformat.rsc"  ;

: title$ ( --- addr ) \ titre de l'access. pour le menu.
  0" Qformat"  ; \ 2 blancs devant titre pour alignement..

 ' qformat  alias  run
include save_acc.scr
 ' runacc Is 'abort

saveacc QFORMAT.ACC

EOF
```

CONTENU DU FORUM 3615 SAM*JEDI

FDI   Du 09.09.88 A [illegible]
CHERCHE UN FORTH POUR ATARI 520 ST TURBO-FORTH CONVIENDRAIT-IL? Y PAS DE TELECHARGEMENT POSSIBLE. DONC TEL 59 20 45 90 MERCI

SECRETAIRE   Du 15.09.88 A 12h46
NOUVEAUTE MODULE 5:
Concernant le MODULE 5 de TURBO-Forth, je signale que le nouveau meta-compilateur peut non seulement creer des programmes sans en-tete, mais aussi generer trois versions de TURBO-Forth:
- en francais
- en anglais
- en allemand

Cette nouveaute mise a la disposition des programmeurs permet de repercuter les corrections, ameliorations et modifications eventuelles sur un seul fichier. Chaque version generee contient en standard trois mots de plus que TURBO de module 1:

```
 ?\      option compil.conditionnelle
 EXIST?  recherche existence de mot
 FRENCH  drapeau linguist.version fran.
 ENGLISH   "        "        "   angl.
 GERMAN    "        "        "  allem.
```

Le programme META.FTH a ete modifie de maniere a fonctionner de la meme maniere sur TURBO M1 que TURBO etendu. Chaque version meta-compilee dispose de son propre identificateur linguistique (FRENCH, GERMAN, ENGLISH). La presence d'un identificateur est analysee par EXIST?: Exemple:

```
  : CHOIX ( --- fl)
  EXIST? FRENCH  ?\ ." choix O/N: "
  EXIST? ENGLISH ?\ ." choice Y/N: "
  EXIST? GERMAN  ?\ ." Wahl J/N: "
  KEY
  EXIST? FRENCH  ?\ ASCII O =
  EXIST? ENGLISH ?\ ASCII Y =
  EXIST? GERMAN  ?\ ASCII J =
  IF TRUE ELSE FALSE THEN ;
```

Cette option peut tres facilement etre adaptee a d'autres langues etrangeres (espagnol, italien, neerlandais, breton, verlan, etc...). Mais elle permet surtout pour le moment de creer un seul programme qui soit trilingue. Chaque version (francais, allem., anglais,...) compilera exclusivement les portions de texte ou de routines le concernant. A l'execution et a la decompilation, exemple avec la version anglaise, le mot CHOIX se lira comme s'il avait ete defini par:

```
  : CHOIX
  KEY ASCII Y =
  IF TRUE ELSE FALSE THEN ;
```

Alors si apres ca on ne dit pas que TURBO s'ouvre a l'Europe, je m'exile au GUATEMALA, car ils ont encore de l'espoir de sortir de leur sous- developpement... sans blague!

LAMBERTPH   Du 15.09.88 A 20h38
URGENT --- INCLURE LA PROGRAMMATION ORIENTEE OBJET TELLE QUE PRESENTEE PAR M PETREMANN (JEDI 45) DANS LE NOYAU DE TURBO-FORTH POUR METAGENERATION --- QU'ELLES SONT LES MODIFS A APPORTER AU METAGENERATEUR POUR QU'IL PRENNE EN COMPTE DES MOTS DE DEFINITION N'EXISTANT PAS DANS LE NOYAU D'ORIGINE ET ACCEPTE LES MOTS CREES A L'AIDE DE CEUX-CI PLUS GENERALEMENT QUI POURRAIT M'EXPLIQUER PAR LE MENU LE FONCTIONNEMENT DU META-GENERATEUR
PHILIPPE LAMBERT - VELIZY 78 - TEL 39 46 14 49 POUR UNE REPONSE RAPIDE MERCI

SECRETAIRE   Du 16.09.88 A 12h55
PERFORMANCES DU META-COMPILATEUR fourni dans le module 5 de TURBO-Forth. On a dit que le compilateur de TURBO-Forth est lent, beaucoup plus que celui de TURBO-Pascal ou C. Mais TURBO-Forth est un interpreteur/compilateur. Ensuite, le code genere est le plus compact qui soit. Pour preuve, voici des exemples d'une meme routine compilee en divers langages et leurs compilateurs associes.

suite en page 15

...texte du précédent article.

FORMATAGE SUR ATARI ST
par Daniel FLIPO

Système: VolksFORTH-83

L'objet de cet article est de présenter un programme de formatage sous GEM. Outre l'intérêt pratique du programme, qui autorise un large choix de formats (9 ou 10 secteurs par piste, choix du facteur d'entrelacement, format ATARI ou MS/DOS ...), les boîtes de dialogue gérées par le programme me paraissent bien illustrer les possibilités du GEM dans ce domaine, qui vont bien au-delà de ce que permet le mot DIALOG présenté dans DEMOGEM.SCR (JEDI N°43).

Enfin les écrans 16 et 17 indiquent comment transformer le mot FORTH QFORMAT, soit en programme .PRG (à double-cliquer) soit en accessoire (dans ce cas le fichier SAVEACC.SCR cf présent numéro est nécessaire).

I) Ce que fait le mot QFORMAT.

QFORMAT propose de formater une disquette simple ou double face, présente sur le lecteur A ou B, en 9 ou 10 secteurs par piste et 80 pistes par face. Le choix (standard) de 9 secteurs donne une capacité de 720 ko environ en double face contre 800 ko pour 10 secteurs.

Le choix entre le formatage standard ATARI ou MS/DOS est proposé, les disquettes obtenues avec l'option MS/DOS sont utilisables indifféremment sur ATARI ou sur compatible IBM. A retenir par ceux qui voudraient envoyer leurs textes sur disquette au secrétaire: il est équipé pour lire les disquettes 3 pouces et demi double face 720 ko... mais au format MS/DOS.

Les différences entre les deux formats sont exposées au III.

QFORMAT donne également le choix du facteur d'entrelacement (interleave en anglais.). Sa valeur détermine normalement l'ordre dans lequel les secteurs du disque sont lus et écrits. Sur l'ATARI, il semble que son rôle soit un peu différent: la valeur standard est 1, ce qui signifie que les tops de changement de piste sont "sur" (en fait juste avant) le 1er secteur de chaque piste. En effet en plus des données, une disquette contient des informations système (ID, GAP..) mises en place par le formatage entre les secteurs de données.

Sur chaque piste il y a place pour 10 secteurs et un petit bout de 11ème inutilisable. Le choix de la valeur 11 pour le facteur d'entrelacement place les tops de changement de piste "sur" le 11ème secteur, ce qui évite à la tête de lecture de faire un tour pour rien à chaque changement de piste: en effet, lorsque ces informations sont en tête de piste la tête de lecture refait un tour complet avant de lire le premier secteur!

Pour un formatage en 9 secteurs par piste ceci permet de diviser le temps de lecture/écriture par 1.5 environ. On entend d'ailleurs très nettement à l'oreille que le déplacement de la tête de lecture est plus rapide.
Attention, ceci n'est vrai que pour les fichiers longs enregistrés d'un seul bloc sur des pistes consécutives, pas pour les fichiers morcelés pour boucher les "trous" laissés par des fichiers effacés.

Pour un formatage en 10 secteurs par pistes, on ne gagne rien à choisir un facteur d'entrelacement différent de 1. Il semble que le lecteur ne parvienne pas à lire d'affilée le 10ème secteur, les informations de changement de piste et le 1er secteur de la piste suivante. Le programme verrouille d'ailleurs ce choix (voir II-1).

La modification du facteur d'entrelacement est SANS AUCUNE CONSEQUENCE sur la sécurité des données ; en particulier la fonction de vérification n'est pas supprimée.

QFORMAT repère les éventuels secteurs défectueux, et enregistre leur place sur la F.A.T. afin qu'aucune donnée ne puisse y être placée ultérieurement, ce qui est INDISPENSABLE et malheureusement omis par bon nombre de programmes de formatage en circulation! La boite d'information finale indique le nombre d'octets disponibles sur le disque et le nombre de secteurs défectueux sur chaque face.

Sur les aspects techniques du formatage (notions de cluster défectueux, de boot-sector, de F.A.T.) on pourra consulter:

- Le livre du lecteur de disquette MICRO APPLICATIONS N°13
- Les articles de François Guillemé sur "La gestion des disquettes" dans les N°13 à 16 de ST-MAGAZINE.

ATTENTION :

1) Le mot QFORMAT ne doit être exécuté qu'après avoir initialisé le GEM par le mot GRINIT, puis chargé le fichier ressources QFORMAT.RSC et réservé la mémoire nécessaire par le mot SETTINGS. Le fichier de ressources QFORMAT.RSC doit être présent sur la disquette et pas à l'intérieur d'un dossier.

2) Cliquer sur FORMATER dans la première boite de dialogue lance le formatage immédiatement; assurez vous que la disquette à formater est bien dans le bon lecteur! Cliquer sur ANNULER fait sortir du programme. Malgré tout, pour les distraits, j'ai choisi de commencer le formatage à partir de la dernière piste (de 79 à 0) pour ne détruire qu'en dernier lieu les informations vitales contenues sur les pistes 0 et 1 (Boot-sector et F.A.T.). On peut ainsi sauver la mise en coupant tout assez vite!!!

II) Fonctionnement des trois boîtes de dialogue.

Les caractéristiques des 3 boîtes illustrant le source sont incluses dans le fichier de ressources QFORMAT.RSC qui doit être crée à partir d'un programme de construction de ressources (RCS de Digital Research, ou K-RESSOURCES).

II-1) Boîte de choix

(voir figure 1)

Les rectangles "DISQUE A", "DISQUE B" etc. sont des objets GEM du type BUTTON, avec les attributs SELECTABLE (rend possible l'affichage en inverse vidéo), RADIO-BUTTON (bascule entre choix qui s'excluent mutuellement), et éventuellement PRESELECT pour ceux qui apparaîssent en inverse vidéo au départ.

Chaque couple de boutons disque A ou B, simple ou double face, etc doit être inclus dans une boite invisible du type IBOX (largeur du bord = 0) afin que la sélection de "disque A" par exemple ne désélectionne que "disque B" et pas les autres boutons!

Les choix à faire ne sont pas toujours indépendants les uns des autres. On peut souhaiter interdire la sélection d'un bouton si telle option a été choisie. Exemple: pour formater au standard MS/DOS il faut interdire l'option 10 secteurs par pistes ainsi que le choix d'un facteur d'entrelacement autre que 1.

Pour cela il faut ajouter l'attribut EXIT au bouton MS/DOS (c'est ce qui fait que son cadre est un peu plus épais que celui des autres...). Ainsi on sortira de la boîte de dialogue lors de la sélection de MS/DOS et c'est le programme qui devra ajouter l'attribut DISABLE aux boutons "10 secteurs" et "facteur: ", selectionner les boutons antagonistes "standard" et réafficher les 4 boutons modifiés. Les boutons "standard" apparaîtront alors en inverse vidéo, et les deux autres en grisé comme sur la figure ci-dessus.

L'élément "Facteur : " n'est pas à proprement parler du type BUTTON mais du type EDIT (texte éditable). Ses attributs sont SELECTABLE, RADIO BUTTON et EDITABLE. C'est ce qui permet à l'utilisateur d'entrer au clavier une valeur de son choix (ici deux chiffres). Il faut utiliser

les touches ESCAPE ou BACKSPACE pour effacer la valeur existante.

L'élément "Nom du disque : " est également du type EDIT avec pour seul attribut EDITABLE. Le nom entré au clavier sera inscrit sur le disque en tête du répertoire principal suivi de l'octet 08 pour indiquer qu'il s'agit du nom du disque et pas d'un nom de fichier (cf les deux dernières lignes du mot CREATE_FAT, écran 14).

"Formater" et "Annuler" sont des objets du type BUTTON avec les attributs SELECTABLE, EXIT, et en plus DEFAULT pour "Annuler" ; ainsi la frappe sur la touche RETURN a le même effet qu'un clic sur "Annuler".

Il est maintenant facile de comprendre le fonctionnement du mot OPTIONS_CHOICE (cf. écran 9) qui gère cette boîte : le mot FORM_DO attend que l'utilisateur clique sur un bouton EXIT, la boite devient alors inactive, mais, si c'est une fausse sortie justifiée par une modification de l'apparence de la boîte, on reste dans la boucle BEGIN ... UNTIL, on modifie l'affichage, puis on réactive la boite par un nouveau FORM_DO.

La constante :name qui est sur la pile à l'appel de FORM_DO indique au GEM que le curseur texte doit se trouver sur l'objet "Nom du disque : " . Pour le faire passer d'un objet de type EDIT à l'autre on utilise les flèches de déplacement haut et bas.

Le mot STORE_OPTIONS reporte les choix faits sur la boîte de dialogue dans les variables ad hoc. La valeur éventuellement entrée au clavier comme facteur d'entrelacement est récupérée par le mot GETNUMBER de SUPERGEM.SCR.

II-2) Boite N°2 : évolution du formatage.

(voir figure 2)

Au fûr et à mesure du formatage la barre hachurée se déplace vers la droite comme lors d'un formatage à partir du bureau GEM.

Comment réaliser cela? C'est simple: la boîte contient sous les mots "Face 1" et "Face 2" deux boites du type BOX dont l'intérieur est blanc qui elles-mêmes contiennent une boite hachurée de mêmes dimensions. Tout objet GEM contenu dans un autre est appelé son fils. Au départ on ne fait afficher que l'objet père (rectangle blanc) en limitant la "profondeur d'affichage" à 1, voir le mot SHOW_INFO écran 7. En choisissant une profondeur 2 on afficherait l'objet père et ses fils mais pas les petits-fils et ainsi de suite...

Ensuite au cours du formatage (cf 4ème ligne du mot FORMAT_SIDE écran 14) on affiche une partie seulement des rectangles hachurés grâce au mot OBJC_DRAW qui demande les coordonnées (x, y, w, h) de la zone autorisée d'affichage (clipping zone).

A la fin du formatage le mot HIDE_INFO fait effacer la boîte de dialogue et restaure l'arrière plan sauvegardé par SHOW_INFO.

II-3) Boite N° 3 : compte rendu du formatage.

(voir figure 3)

C'est une boîte de dialogue simple gérée par le mot DIALOG dont le fonctionnement a été décrit dans DEMOGEM.SCR (JEDI N°43).

Les nombres d'octets libres et de secteurs défectueux sont installés dans la boite par les mots PUTNUMBER et PUTSTRING définis dans SUPERGEM.SCR.

III) Différences entre les formats ATARI et MS/DOS (version 3.2)

- Le premier octet de chacune des 2 F.A.T. vaut $F7 sur ATARI, contre $F8 pour une disquette simple face 80 pistes (360 ko) ou $F9 pour une double-face 80 pistes (720 ko).

- Chaque F.A.T. occupe 5 secteurs sur ATARI (secteurs 1 à 5 et 6 à 10) contre 3 sous MS/DOS (secteurs 1 à 3 et 4 à 6). Le répertoire principal, qui suit la 2ème F.A.T., commence donc au secteur 11 sur ATARI et 7 sous MS/DOS.

- Sur ATARI, le nom de la disquette est inscrit en tête du répertoire sur 11 octets suivi d'un octet de valeur 08 (lorsqu'un nom est donné au formatage). Sous MS/DOS ce nom est inscrit ...???

- Le dernier mot (16bits) du BOOT-SECTOR (secteur 0 de la disquette, offset = $1FE) doit valoir obligatoirement $55AA sous MS/DOS, alors que sur ATARI ce mot contient une valeur appelée CHECKSUM égale à la somme des mots du BOOT-SECTOR. D'autre part, le nombre de secteurs occupés par chaque F.A.T. est inscrit dans le vingt-troisième octet (offset = $16) du BOOT-SECTOR. Cet octet doit donc contenir la valeur 3 sous MS/DOS et 5 sur ATARI.

■

Suite de la page 13

TURBO-Forth et module M5:

```
  : HELLO 10 EMIT 13 EMIT
    ." Hello, world" BYE ;
```

programme genere: HELLO.COM 0.518 Ko. Moins de 1K!!!

Turbo-PASCAL:

```
  program hello;
  begin writeln("Hello, world") end.
```

Programme genere: HELLO.EXE 1,857 Ko

TURBO-C:
(listing tres long, car beaucoup d'appels de bibliotheque)
Programme genere: HELLO.EXE 5,734 Ko

A titre anecdotique, dBASEIII+:

```
  ? "Hello, world"
  QUIT
```

Programme compile par CLIPPER, fichier genere: HELLO.EXE 116.784 Ko (arghhh!!!).

Le plus rapide: TURBO-Pascal 4.0
Le plus compact: TURBO-Forth
Le plus verbeux: TURBO-C

Temps de compilation, plus rapide au moins rapide: TURBO-PASCAL, TURBO-C, TURBO-Forth, dBASEIII+CLIPPER.

Seuls TURBO-Forth et TURBO-Pascal ne generent pas de fichier intermediaire OBJ

SECRETAIRE Du 16.09.88 A 13h02
REPONSES A LAMBERTPH:
Enfin la question a 1000 FR: Non, on n'envisage pas d'inclure la programmation en langage oriente objet dans le noyau de TURBO-Forth. En fait, il n'y aura JAMAIS de version 2 ou 3 ou 4 de TURBO-Forth. A notre avis, c'est au programme utilisateur d'apporter son extension, modification ou complement. J'aurai l'occasion de m'etendre plus longuement sur cette opinion dans le cadre d'un article a paraitre dans JEDI. Maintenant, concernant la prise en compte des mots de definition par le meta-compilateur, il faut:
- definir le mot dans l'hote,
- definir dans la cible la partie execution (style DOVAR, DOCONST, DOSTRING, etc...).

Cette technique et son mecanisme est tellement subtile qu'elle ne peut faire l'objet d'une reponse sommaire dans ce FORUM. Donc, reponse dans JEDI.

FORTH7 Du 17.09.88 A 19h57
Concernant le Forth Oriente Objet: le sujet est a la mode. Le dernier numero de Forth Dimensions y est totalement

Suite page 24

Journal of Forth Application and Research Volume 4, Number 4

Use of a Forth-Based Prolog for Real-Time Expert Systems

I. Spacelab Life Sciences Experiment Application

*William H. Paloski, Louis L. Odette,**
Alfred J. Krever,† and Allison K. West

KRUG International
Technology Life Sciences Division
17625 El Camino Real, Suite 311
Houston, TX 77058

Abstract

A real-time expert system is being developed to serve as the astronaut interface for a series of Spacelab vestibular experiments. This expert system is written in a version of Prolog that is itself written in Forth. The Prolog contains a predicate that can be used to execute Forth definitions; thus, the Forth becomes an embedded real-time operating system within the Prolog programming environment. The expert system consists of a data base containing detailed operational instructions for each experiment, a rule base containing Prolog clauses used to determine the next step in an experiment sequence, and a procedure base containing Prolog goals formed from real-time routines coded in Forth. In this paper, we demonstrate and describe the techniques and considerations used to develop this real-time expert system, and we conclude that Forth-based Prolog provides a viable implementation vehicle for this and similar applications.

Introduction

The recent flurry of activity in commercial expert system development has all but bypassed the real-time computing community. Although it may be desirable to incorporate expert systems in certain real-time computing problems, the amount of computing over symbols (reasoning) required by an expert system is difficult to implement in real time. Indeed, it is often difficult to implement the amount of computing over numbers required by real-time applications. The commercial artificial intelligence (AI) industry is moving slowly toward providing fully formed products for real-time expert system development (cf. [WOL87], [LEI87]); however, general real-time symbolic processing remains an AI research problem. It may be several years before AI languages and development/delivery environments can play a significant role in real-time applications.

Owing to this lack of suitable, commercially available real-time expert system development vehicles and to its interest in developing expert systems for data acquisition and control applications, the Forth community has recently become active in developing real-time expert systems [PAR86]. Some of the contributions of this community in this area include a diesel electric locomotive repair expert system [JOH83], an orbiting spacecraft command anc l expert system [HAR86], a spacecraft trajectory processing data error detection expert sy RAPS) [RAS86], a real-time polysomnographer expert system (FORTES) [RED87], a sleep .er diagnosis system [TRE87], some general real-time expert system development packages (EXPERT-2 [PAR84], FORPS [MAT87]), and a real-time version of the common expert system development language OPS5 (REAL-OPS) [DRE86]. In each of these applications, a set of high-level (AI) programming tools (i.e., inference engine, language parser, etc.) is built from Forth primitives to take advantage of the high run-time execution speed offered by Forth.

In the current paper, we describe a new Forth-based real-time expert system. This application is being developed using a unique implementation of the logic programming language Prolog, in which the Prolog interpreter is embedded in a Forth environment [ODE87]. This Prolog is fully compatible with the de facto standard described by Clocksin and Mellish [CLO81] and provides a simple and direct interface to the underlying Forth. This interface allows creation of Prolog goals from Forth words and provides the basis for rapid development and integration of the real-time routines into an expert system knowledge base. Real-time algorithms stored in the knowledge base through this mechanism can subsequently be executed on a logic-driven basis by the expert system.

The real-time expert system presented here will be used to assist astronauts in performing a series of life sciences experiments aboard the First International Microgravity Laboratory (IML-1) Spacelab mission. It will serve as the operator interface to the experiment data acquisition and control system. The series of experiments to be controlled by the expert system is known as the Microgravity Vestibular Investigations (MVI) and is designed to study the role of the inner ear in space motion sickness. An expert system was considered for the MVI experiments because of the complexity of the experimental procedures, the suboptimal experiment environment provided by Spacelab, and the high cost of failure. Highlights of this application are presented to demonstrate our general approach to building real-time expert systems using the Forth-based Prolog.

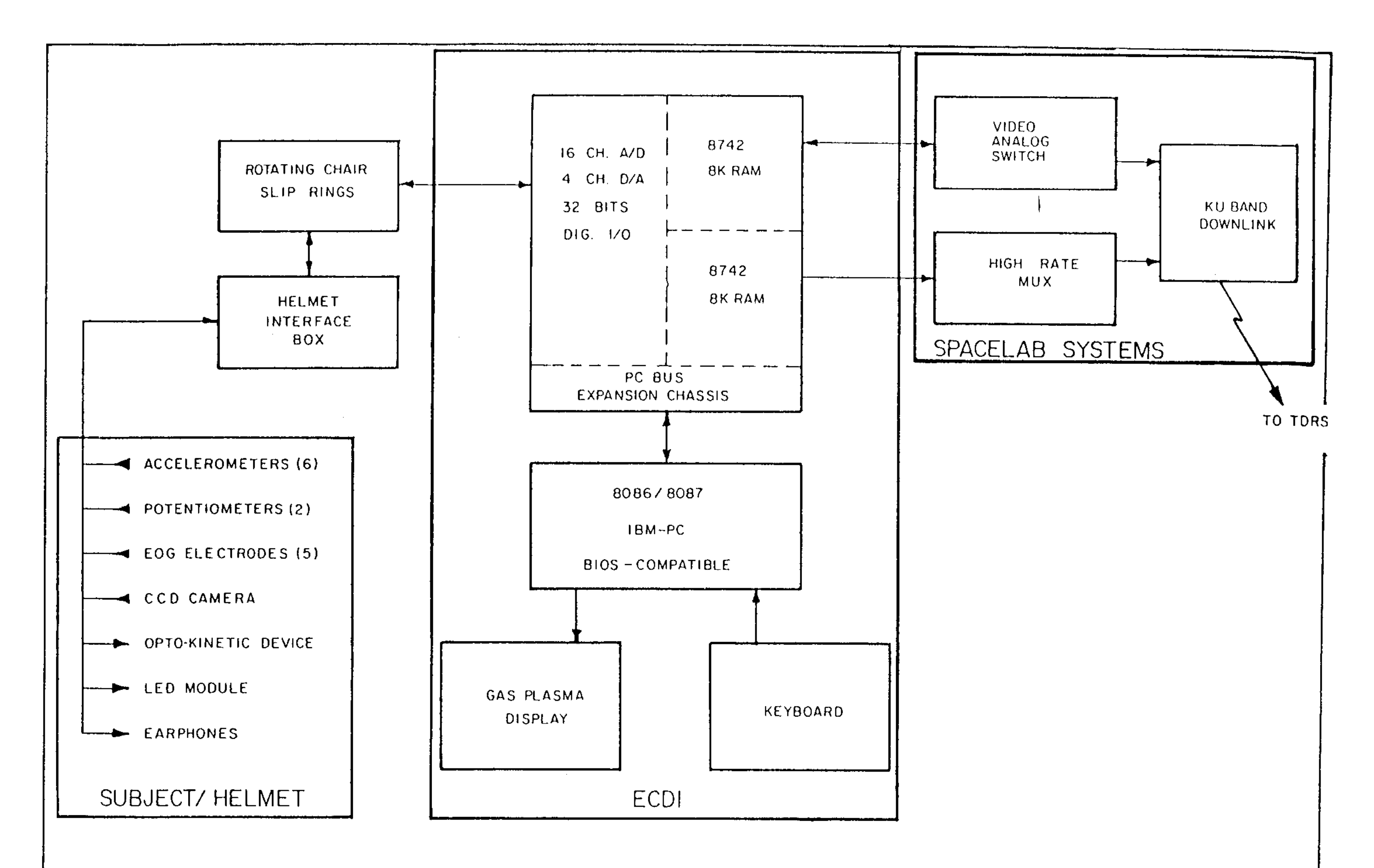


Figure 1
Block diagram of MVI hardware.

Instrumentation

The MVI experiments comprise six separate scientific functional objectives, each of which is addressed by performing 3 to 24 related experiments. These experiments will be performed during the IML-1 mission by teams of two astronauts. One astronaut will serve as the experiment subject. He will be strapped into a rotating chair mounted in the Spacelab center aisle and will don a helmet supporting various physical and physiological sensors. Throughout the experiments, he will be presented with inertial, auditory, and/or visual stimuli, and his physiological responses to those stimuli will be monitored, graphically displayed, and transmitted to Earth. The other astronaut will serve as the experiment operator. He will be responsible for setting up and calibrating each of the sensors and for performing each step of the 65 experiments. During each experiment, he will also monitor the response, safety, and well-being of the subject and communicate with ground-based scientific investigators.

The experiments will be controlled by the astronaut/operator using a rack-mounted console known as the Experiment Control and Data Interface (ECDI; Figure 1). The heart of this console is an IBM PC-compatible, 8086-based, laptop microcomputer (Grid Case III, Grid Systems Corporation, Mountain View, California). Within the ECDI, the PC bus is extended to a 9-slot expansion chassis (IBUS Systems Corporation, San Diego, California) which houses all the data acquisition, control, and Spacelab system interface hardware. Data acquisition and control capabilities are provided by two analog interface boards (DT2801, Data Translation, Marlborough, Massachusetts) containing a total of sixteen 12-bit differential channels of analog-to-digital (A/D) conversion, four 12-bit channels of digital-to-analog (D/A) conversion, and 32 bits of digital input/output (I/O) lines. Analog data from electro-oculogram (EOG), head acceleration, and head position sensors attached to the subject and helmet will be sampled at either 128 Hz (5 channels) or 32 Hz (10 channels) and digitized using the A/D converters. Command signals, issued through the D/A converters and digital I/O ports, will control an optokinetic device, a light emit diode (LED) array, ear phones, and other experiment stimulus-producing devices.

Following digitization, two channels of experiment sensor data will be graphically displayed in real time on the ECDI computer screen. In addition, all acquired data, as well as all command and experiment condition data, will be serially transmitted at 25.6 Kbits/sec to Spacelab downlink systems. Spacelab system interfaces are provided by two custom boards containing I/O-mapped, 8742 Universal Peripheral Interface processors. These boards will control data transfer between the experiment computer and either the Spacelab high rate multiplexer (HRM) or the Spacelab video analog switch (VAS) using pairs of memory-mapped RAM ping-pong buffers and Universal Asynchronous Receiver/Transmitters. The HRM and VAS provide interfaces for digital and video data telemetry to ground-based scientific and mission control observers.

Expert System

Although Spacelab provides a unique facility for studying the effects of microgravity on various physical and physiological processes, its environment is less than optimal for performing complex scientific experiments. Crew time and training are limited, communication between astronaut/operators and ground-based scientific investigators is limited, and crew members are likely to experience some degree of motion sickness during the first few days of a mission. The MVI expert system is designed to reduce the impact of these factors on the quality of the MVI data collection. This expert system should enhance the ability of the astronaut/operator to collect high quality data in the Spacelab environment by managing the complexity of the experimental procedures and by offering guidance through the experiment operations.

The heart of the expert system is its knowledge base, which is made up of a data base containing facts about experiment procedures and protocols, a rule base containing general rules for carrying out the experiments under normal and abnormal conditions, and a procedure base containing real-time routines coded in Forth. The rule base is used to step the operator through each experimental protocol, as specified in the data base, while simultaneously activating appropriate Forth tasks stored in the procedure base. The knowledge base is primarily declarative, relying on the Prolog inference engine for control; however, the Prolog **builtin** predicate, used to incorporate Forth routines into the knowledge base, allows procedures to be invoked from Prolog (a complete description of the Forth-based Prolog is provided in a companion paper in this issue [ODE87]).

A knowledge base structure was selected for representing the experiment procedure data rules in order to provide a flexible system that could be easily modified. This format allows separation of the data from the inference rules and real-time routines and thereby permits restructuring of rules without changing the data organization or real-time behavior. The structure of a knowledge base also greatly reduces the distance between the program and the user; both data and rules are stored in blocks of text that can be read as paragraphs by natural language processing routines as well as by the programmer and the expert Principal Investigators. This feature will aid in the development of user-friendly I/O routines and explanation facilities.

There is a top-level query language through which the experiment operator has access to the rule base. Upon being queried, the rule base extracts appropriate data from the data base and provides that data as input to real-time algorithms in the procedure base. The real-time algorithms then use this data in setup and execution of machine control tasks. At the present time, the data and procedure base development is essentially complete. The rule base remains under development; however, all of its essential components, including those described below, have been developed and tested. A detailed description of the knowledge base and execution process follows.

Data Base

The MVI expert system data base consists of six records, each of which encodes all of the operational requirements for a single functional objective. Each record ha… structure (Figure 2). The primary node in each record is the functional objective identifier …f the n secondary nodes branching from the primary node identifies the location of all th… …ation required to

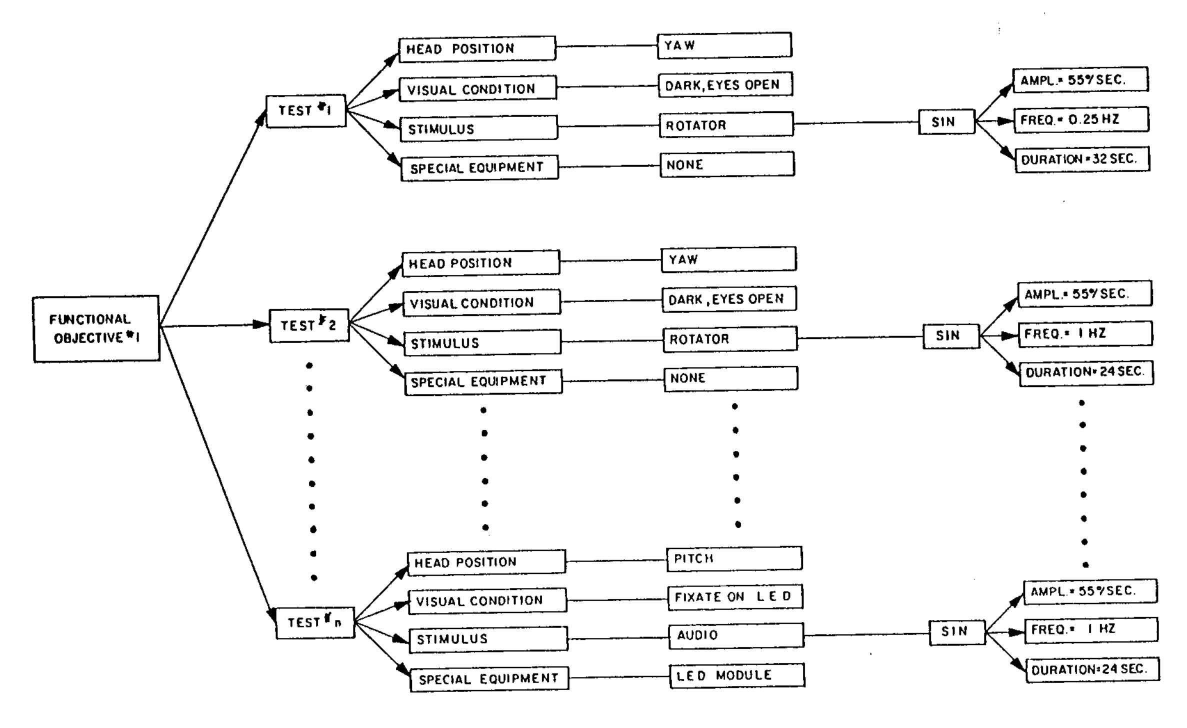


Figure 2
Example of the tree structure of the expert system data base.

perform a single experimental protocol (test) within the functional objective. Four tertiary nodes branching from each secondary node identify the location of the head position, visual condition, experimental stimulus, and special equipment information required by the particular protocol. The set of nodes branching from the tertiary nodes contains the actual data pointed to by the identifiers. All nodes further to the right add clarification and refinement to this data. As an example, Figure 2 shows that the experimental protocol Test #1 of Functional Objective #1 requires that an inertial stimulus be provided by the rotating chair and that this stimulus have a sinusoidal velocity profile with an amplitude of 55°/sec, a frequency of 0.25 Hz, and a duration of 32 seconds.

The Prolog clause required to encode a data tree similar to that diagrammed in Figure 2 is presented in Figure 3. Each branch node is represented by a predicate having an arity equal to the number of parenthetical objects following it and separated by commas. The arity of a predicate is fixed. For example, the arity of the predicate **functional_objective** is 3: the object **1** (the functional objective number); the object **vor_suppression** (the functional objective name); and a list, delimited by square brackets, which contains objects, predicates, and other lists that identify the remainder of the data required for Functional Objective #1. In Prolog, records can be organized into pseudo-English paragraph structures like that in Figure 3. This helps to bring the source code much closer to both the programmer and the domain expert. Prolog unification is the mechanism used by the expert system to access data fields within the data base.

```
146 LIST
     /* expert system development area */
     functional_objective(1,vor_suppression,
     [test(1,head_position(yaw),
             visual_condition(d,[dark,eyes_open,straight_ahead]),
             stimulus(rotator(sin(ampl(55),freq(25),dur(32)))),
             special_equipment(none),
      test(2,head_position(yaw),
             visual_condition(d,[dark,eyes_open,straight_ahead]),
             stimulus(rotator(sin(ampl(55),freq(1),dur(24)))),
             special_equipment(none)),
      test(3,head_position(pitch),
             visual_condition(f1,[fixate_on_LED_module_target]),
             stimulus(audio(sin(ampl(55),freq(1),dur(24)))),
             special_equipment(LED_module))]).
```

Figure 3

Prolog clause required to encode a data base record similar to the example presented in Figure 2.

Rule Base

The rule base will contain approximately 50 Prolog clauses, which the operator accesses via the user query language and the expert system accesses through unification. The primary execution rules have been developed, and some of them appear in Figure 4. Lines 2-3 in Block 147 present the general rule for executing a functional objective. The operator can execute a functional objective using the query **fo(X)**. If **X** is instantiated (equivalenced) to a functional objective number for which a data record exists within the expert system data base, then that particular functional objective will be executed. If **X** is instantiated to a functional objective number for which no data record exists, the second **fo(X)** clause (Block 147, Line 5) will be called to report the error. If **X** is not instantiated, the first functional objective record in the data base will be found and used for execution. Thereafter, successive functional objectives could be executed by forcing Prolog backtracking at the conclusion of each functional objective execution. This could be accomplished by requesting more solutions to the top-level query.

```
147 LIST
 0  /*  expert system development cont  */
 1
 2  fo(X) :- functional_objective(X,Y,L),!,
 3          execute_fo(L).
 4
 5  fo(X) :- nl,nl,write(invalid_fo_number),nl.
 6
 7
 8  execute_fo(L) :- member(test(N,HP,VC,ST,SE),L),
 9                   perform_test(N,HP,VC,ST,SE).
10
11
12  execute fo(L) :- nl,nl,write(fo_complete),nl.
13
14
15
```

```
148 LIST
/*  expert system development cont  */

perform_test(N, head_position(Pos), visual_condition(Eyes,_),
            stimulus(Input_Device), special_equipment(Device)) :-

                set_up(Device),
                restrain_head(Pos),
                command_subject(Eyes),
                data_on,
                start(Input_Device),
                stop(Input_Device),
                data_off,
                command_subject(relax).
```

Figure 4

Prolog source code implementation of a portion of the MVI expert system rule base.

Procedure Base

The procedure base is a set of Forth colon and code definitions used to control all the MVI experiment hardware and to execute real-time data acquisition, display, and transmission tasks. Sample code demonstrating the mechanism used for building the procedure base is presented in Figure 5. Block 141 is a Forth block, compiled using the Forth `LOAD` command. Block 145 is a Prolog block (as are all blocks in Figures 3 and 4) and is compiled using the Forth word `CONSULTING` (see Block 141, Line 11), which has an action similar to that of the Prolog predicate **consult** [CLO81].

In Block 141, a small polyFORTH terminal task, `SIMUL`, is constructed in Line 1 [VAN83]. A simple function is assigned to that task: the Forth word `SEE` (Line 4) causes the contents of the variable `EVENTS` to be displayed on the user's terminal and increments the displayed value twice per second; the Forth word `DONE` (Line 5) deactivates the background task. In Lines 7 and 8, two new Forth words, `$DATA_ON` and `$DATA_OFF`, are defined. These words are formatted to become Prolog built-in predicates; they include the `R> DROP` necessary for execution in the Prolog environment [ODE87] and a `$TRUE` word following all other execution, which indicates to Prolog that the word succeeds (is true). `$DATA_ON` starts the terminal task and `$DATA_OFF` stops the terminal task. The Forth words `$DATA_ON` and `$DATA_OFF` are defined as Prolog objects **data_on** and **data_off** in Block 145 (Lines 4 and 5). These objects can be used as Prolog goals, as can any other Prolog predicate; however, the process of testing one of these goals results in execution of its Forth procedure. Forth `CODE` definitions can be made Prolog objects using the same technique.

```
141 LIST
 0  ( Simple multitasking example)
 1  500 64 0 TERMINAL SIMUL     GILD   SIMUL CONSTRUCT
 2
 3  VARIABLE EVENTS
 4  : SEE     SIMUL ACTIVATE BEGIN 1 EVENTS +! 500 MS AGAIN ;
 5  : DONE    SIMUL ACTIVATE STOP ;
 6
 7  : $DATA_ON    R> DROP  ." Start" CR  SEE  $TRUE ;
 8  : $DATA_OFF    R> DROP  ." STOP" CR  DONE  $TRUE ;
 9
10
11  145 148 CONSULTING  ( application rule base follows)
12
....
145 LIST
 0  /* test screens for prolog -- needs SIMUL task */
 1  member(X,[X,_ ]).
 2  member(X,[_,Y]) :- member(X,Y).
 3
 4  data_on :- builtin($DATA_ON) .
 5  data_off :- builtin($DATA_OFF) .
 6
 7...
```

Fig 5.

Logic-Driven Real-Time Procedures

The main advantage of this real-time expert system is its ability to perform real-time data acquisition and control tasks on a logic-driven basis. To perform Functional Objective #1 of the MVI experiments, for example, the operator could type in **fo(1)** at the ECDI keyboard. This would cause Prolog to locate the first **fo(X)** rule (Figure 4, Block 147, Line 2), instantiate the **X** to 1, and attempt to prove the rule true by satisfying all the goals on its right-hand side. To satisfy the final goal **execute_fo(L)**, Prolog would locate the first **execute_fo(L)** rule (Block 147, Line 8) and attempt to satisfy each of the goals on its right-hand side. The first of these goals, **member(test(N, HP, VC, ST, SE), L)**, identifies the parameters of the first test in the data base record associated with Functional Objective #1 (Figure 3) using the member rules given in Block 145, Figure 5. The second of these goals, **perform_test(N, HP, VC, ST, SE)**, causes Prolog to perform that test using the **perform_test** rule in Block 148. Prolog performs the test as a side effect of trying to prove the **perform_test** goal true. Unification is used to pass parameters found in the data base record into the procedure base.

The goals in the body of the **perform_test** clause sequentially perform a generic vestibular experiment. For example, using the Prolog execution logic (cf. [CLO81]) and the data base record presented in Figure 3, the variable **Device** would be instantiated to **none** in the **perform_test** rule. The first goal in the body of that rule would then become **set_up(none)**, indicating that no special equipment is required to perform this particular test. The operator would be informed of this fact when Prolog attempted to satisfy that goal (code not shown). As the Prolog execution continued, the operator would next be instructed to restrain the subject's head in the yaw position, then to be sure that the subject's eyes are in the dark, and finally to command the subject to keep his eyes open and look straight ahead (code not shown).

Next, having completed the setup phase of the test, the expert system would begin performing the test. To do this, Prolog would attempt to satisfy the goal **data_on**, which is a built-in Forth definition (described previously). The Forth definition `$DATA_ON` would be executed as Prolog attempted to determine whether the goal **data_on** is true (the last Forth word in the `$DATA_ON` definition is `$TRUE` which indicates to Prolog that the goal has succeeded). Thus, in satisfying the **data_on** goal, Prolog indirectly executes the Forth definition `SEE`. In the current example, `SEE` activates the background task `SIMUL` (see **Procedure Base**). In the MVI expert system, however, the **data_on** goal activates a Forth background task that acquires analog data from the subject, graphically displays selected data channels on the computer screen, and transmits the acquired data into the Spacelab downlink systems. Once the data acquisition, display, and transmission systems had been activated, the expert system would initiate the experiment stimulus. This would occur as Prolog attempted to satisfy the next goal in the body (**start(Input_Device)**) and indirectly activated, in the MVI expert system, another Forth background task, which would control that device according to parameters passed to the task from the data base. Once the stimulus provided by the input device was complete, the **start(Input_Device)** goal would succeed. The expert system would then turn off the device control background task as Prolog satisfied the **stop(Input_Device)** goal and the data acquisition task as Prolog satisfied the **data_off** goal. Finally, the operator would be notified that the test was complete and would be instructed to allow the subject to relax as Prolog satisfied the **command_subject(relax)** goal.

At this point, the expert system would have completed performing the experiment. Prolog would have satisfied **perform_test**, which, in turn, would satisfy **execute_fo**, which is the last goal of **fo(1)**. Prolog would therefore notify the operator that **fo(1)** was complete and would await further instruction. The operator would then have the option of accepting this solution (thereby concluding the experiment) or of asking Prolog to find a new solution, should one exist. The latter option would cause Prolog to backtrack (see [CLO81]) to attempt to resatisfy the initial goal. The Figure 4 rule base is set up so that backtracking would result in an attempted resatisfaction of the **execute_fo(L)** goal. Prolog would successfully resatisfy this goal if it found another test (one not previously found) on the list of tests associated with Functional Objective #1. If such a test were found, it would be executed using the logic presented above (only the parameter values would change). Thus, following each test in the functional objective, the operator would be given the option of proceeding with the next test or aborting the study. By this mechanism, a single set of Prolog rules (Figure 4) can be used to carry out every test under all functional objectives. If the operator chooses to backtrack to the next test after all tests for a particular functional objective have been performed, the **execute_fo(L)** goal would fail. This failure would cause the **fo(1)** goal to fail and Prolog to await a new input goal from the operator.

Real-Time Operation

The MVI expert system provides real-time experiment control by activating and deactivating polyFORTH background and terminal tasks [VAN83] on a logic-driven basis. A portion of the MVI round robin multitasking loop is presented in Figure 6. Prolog (and the expert system) reside in the `OPERATOR` task, which is the primary terminal task in the round robin loop; all keyboard entries are handled through this task. When the MVI computer is turned on, the only active task is `OPERATOR`. The expert system controls the graphics display terminal task and the device driver background tasks as side effects of responding to operator queries (see the previous section). Only those tasks that need to be active at any point in the experiment protocol are activated. Once a task has completed its required action it is deactivated.

To maintain accurate, high rate (128 Hz) data sampling intervals, data acquisition is controlled by an interrupt service routine (ISR) triggered by a Spacelab clock signal. The data transmission background task is controlled by the ISR to assure synchronization between the MVI data stream and the Spacelab downlink system. The expert system can control data acquisition and transmission by masking and unmasking the Spacelab clock interrupt signal and by changing the ISR in use by altering its vector address. Either of these techniques can be employed on a logic-driven basis. By using this multitasking technique, various real-time processes can be executed concurrently with the expert system. This reduces the required logical inference processing rate of the Prolog.

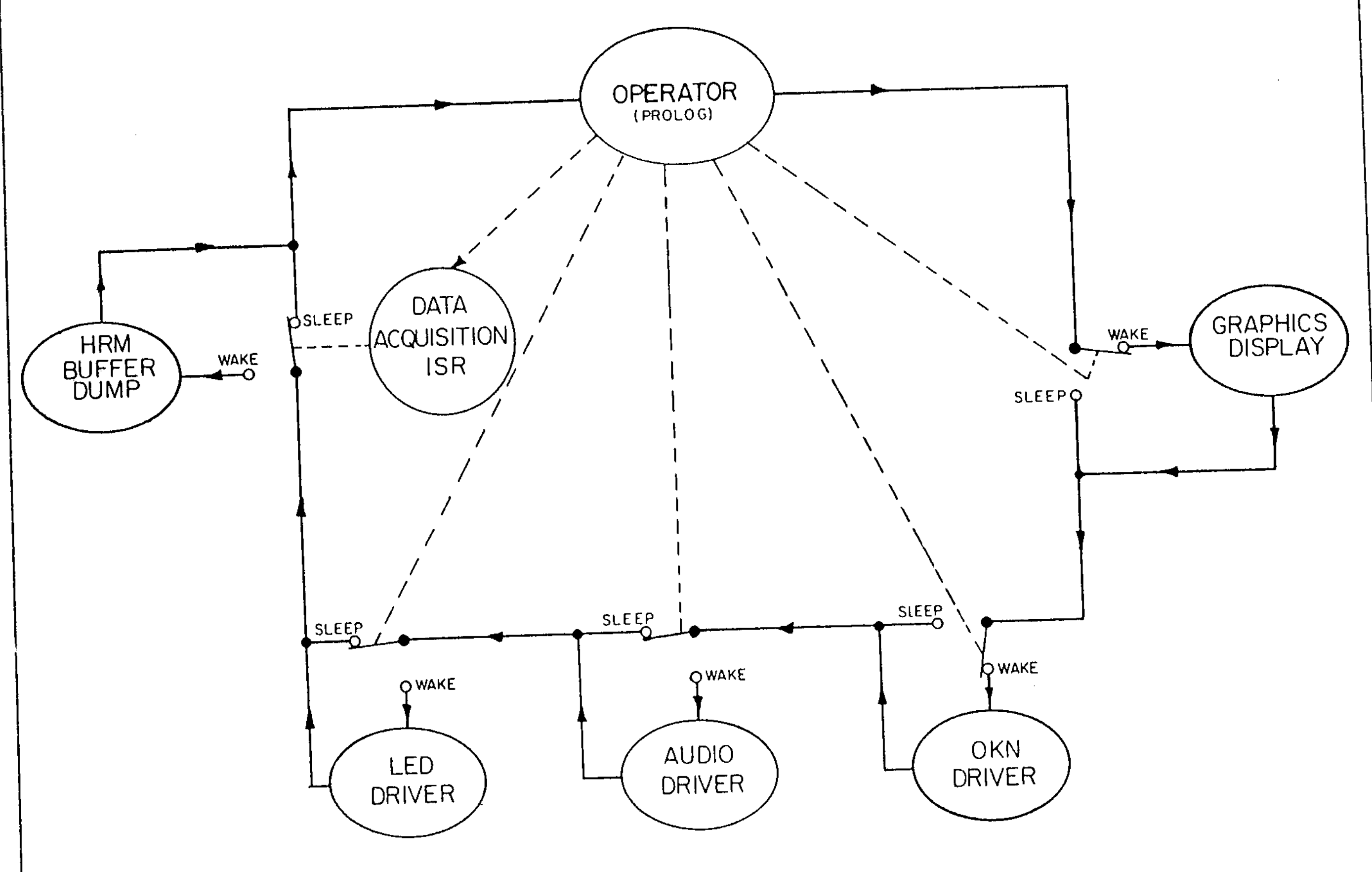


Figure 6
Portion of the MVI expert system round robin multitasking loop.

Discussion

Forth is one of several possible development languages for implementing real-time data acquisition and control processes. We chose it because of its speed, small size, and ability to support rapid prototyping. Prolog is one of several languages for symbolic computation. We chose to use it for several reasons. One significant factor was our familiarity with it. We have several years experience building significant expert system shells and applications (cf. [BAR85]); by implementing Prolog we could reuse a good portion of this software. Another important factor was that Prolog has a small standard kernel; thus, its implementation could be made small and could be accomplished quickly. In our estimation, this compactness is in contrast to LISP; few have braved attempts at a full implementation of the Common LISP standard. Another selection factor was that there is a large community of Prolog users and programmers as well as a sizable body of literature describing Prolog solutions to problems in symbolic computation. The availability of this Prolog literature will allow us to avoid both the reinvention of the wheel and the problems inherent in translating programs. The popular production rule language, OPS5, shares some of these features. Indeed, Dress has developed a real-time version of OPS5 using Forth [DRE86]. Unfortunately, the data-directed style of OPS5 was not a good fit with the goal-directed applications we had in mind. Other Forth-based expert system development tools, such as Expert-2 [PAR84], FORPS [MAT87], and FORTES [RED87], offer smaller kernels and higher execution speeds than our Prolog interpreter but suffer from limited usage, nonstandard syntax, and lack of tested problem-solution paradigms.

We feel that our integration of Prolog and Forth has been successful in addressing a number of important design issues. As the controller for the series of vestibular investigations aboard the IML-1 Spacelab mission, our system must observe close tolerances on the timeline and remain well integrated with the other systems of people and machines on the shuttle. It is therefore imperative that the system provide clear and understandable information to the operator and attempt to reduce operator fatigue and error when on-line. Our approach has been to provide the system with the knowledge it needs to manage these tasks in a variety of situations while simultaneously providing

a flexible user interface that allows the operator to exert greater control if conditions warrant. Prolog, being goal-directed and largely declarative, is well suited to this approach and has allowed us to design and implement rapidly.

Another advantage of our Forth-based Prolog is that the data acquisition, control, display, and transmission procedures are readily coded in Forth, permitting us to satisfy speed constraints. By using a multitasking Forth, we were able to implement an architecture that allowed us to avoid having to queue data or delay responding to changes until reasoning stops—in effect permitting asynchronous reasoning where needed. Indeed, the most unique feature of the MVI expert system knowledge base is its procedure base, which comprises all the Prolog objects formed from Forth colon and code definitions using the **builtin** predicate. By constructing this procedure base, all real-time routines could be developed using Forth and yet controlled by the inference mechanism provided by Prolog. This not only separated software development requirements, but also separated execution speed requirements. Consequently, our real-time expert system could be developed for a small computer with a relatively slow version of Prolog.

The procedure base allows the expert system to manage the real-time computing tasks and thereby reduces the complexity of the procedures that the operator needs to perform. Given the suboptimal environment that Spacelab provides for complex scientific experiments and the high cost of losing data from these experiments, this feature is extremely important. The data and rule bases also contribute to reducing the complexity of the operator's task by "knowing" the experimental protocols, equipment setup requirements, and calibration procedures, and by being able to guide the operator through these.

Every real-time computing task does not require an expert system; in fact, few do. As the complexity of the user interface requirements or software logic increases, however, expert system tools should be considered. We feel that our decision to build an expert system to control the MVI experiments reduced the software development effort. Although many of the features provided by the expert system could have been developed using traditional engineering languages such as Forth, Assembler, or Fortran, the initial program development time would have been much longer, and the software modification and maintenance tasks would have become very difficult. Prolog, by its declarative nature, substantially reduced the distance between the programmer (and expert) and the code. This feature allowed rapid prototyping and simplified software maintenance. Modifications to Prolog source code can often be made by life scientists with no computer training!

Successful space-based computing applications often test the engineering skills of their developers. In many cases, a careful consideration of a host of software/hardware performance issues has led to the choice of Forth as the language for development and delivery ([HAR85], [RAS86], [HAR86]). We have shown that the technology exists today to provide viable knowledge system solutions to such applications as an adjunct to the underlying Forth. Furthermore, we believe that our Forth-based Prolog can be used to provide AI solutions to many well-chosen real-time problems.

Acknowledgement

The authors wish to thank Martin Tracy of FORTH, Inc., for his assistance in implementing a polyFORTH version of Prolog, and Carol Verrett for preparing the manuscript. W. H. Paloski, A. J. Krever, and A. K. West were supported by NASA Contract NAS9-17200.

References

[BAR85] Bartoletti, D. C., Lewis, C. S., Paloski, W. H., Odette, L. L. and Yestin, N. S. 1985. Design of a cardiovascular drug knowledge-base for use in critical care monitoring. *Proc. 11th Ann. NorthEast Bioengineering Conference*. Silver Spring, MD: IEEE.

[CLO81] Clocksin, W. F., and Mellish, C. S. 1981. *Programming in Prolog*. Berlin: Springer-Verlag.

[DRE86] Dress, W. B. 1986. REAL-OPS: A real-time engineering applications language for writing expert systems. *J. Forth Appl. and Res.* 4(2):113-24.

[HAR85] Harris, H. M. 1985. Forth as the basis for an integrated operations environment for a space shuttle scientific experiment. *J. Forth Appl. and Res.* 3(2):23-34.

[HAR86] Harris, H. M. The development of an expert system for the command and control of an orbiting spacecraft. *J. Forth Appl. and Res.* 4(2):305.

[JOH83] Johnson, H. E., and Bonissone, P. B. 1983. Expert system for diesel electric locomotive repair. *J. Forth Appl. and Res.* 1(1):7-16.

[LEI87] Leinweber, D. 1987. Expert systems in space. *IEEE Expert* 2(1):26-36.

[MAT86] Matheus, C. J. 1986. The internals of FORPS: A Forth-based production system. *J. Forth Appl. and Res.* 4(1):7-28.

[ODE87] Odette, L. L., and Paloski, W. H. 1987. Use of a Forth-based Prolog for real-time expert systems. II. A full Prolog interpreter embedded in Forth. *J. Forth Appl. and Res.*, this issue.

[PAR84] Park, J. 1984. *Forth expert system*. Mountain View, CA: Mountain View Press.

[PAR86] Park, J. 1986. Expert systems in Forth. *J. Forth Appl. and Res.* 4(1):3-6.

[RAS86] Rash, J. 1986. Prototype expert system in OPS5 for data error detection. *J. Forth Appl. and Res.* 4(2):297-300.

[RED86] Redington, D. 1986. A Forth oriented real-time expert system: A FORTES polysomnographer. *J. Forth Appl. and Res.* 4(1):47-56.

[TRE86] Trelease, R. B. 1986. Implementation of an experimental microcomputer-based medical diagnosis system. *J. Forth Appl. and Res.* 4(1):57-66.

[VAN83] Van DeWalker, R., and Rather, E. D. 1983. *polyFORTH II reference manual*. 4th ed. Hermosa Beach, CA: FORTH, Inc.

[WOL87] Wolfe, A. 1987. An easier way to build a real-time expert system. *Electronics* 60(5):71-3.

Suite de la page 15

consacre. Turbo-Forth dispose en ce domaine d'atouts majeurs. Des prototypes existent deja, tres avances. nous attendons que les choses se decantent aux US pour sortir un module additionnel O.O. pour TF: nous ne voudrions pas faire cavaliers seuls dans un domaine qui cherche encore ses marques. A vrai dire nous avons meme de l'avance sur les americains. JEDI ne devrait pas tarder a s'en faire l'echo.

Les preoccupations sont communes: la mise au point des outils de developpement de F32 sera considerablement facilitee par l'introduction des concepts de la programmation orientee objets. Quoi de plus "objet" qu'un processeur dont le jeu d'instructions forme les "messages" au sens L.O.O.? Et qu'est-ce qu'un emulateur ou un cross-compilateur sinon des modules d'instanciation ? Suis sincerement desole de n'etre pas a Paris pour la reunion F32. Compte sur notre secretaire pour me remplacer. F7

SECRETAIRE Du 20.09.88 A 08h42
REPONSE A NICOLAS B: Pour telecharger sur SAM*JEDI a l'aide de PROCOMM, utilisez le transfert de fichiers en ASCII. Placez PROCOMM en mode capture, selectionnez le fichier a transferer sur SAM* puis lancez la capture sur PC, tapez ENVOI sur minitel.

ESTHETE Du 22.09.88 A 18h59
AYANT DU MAL A ME LANCER JE CHERCHE BIBLIOGRAPHIE OU TROUVER 'TOUR DE FORTH' MEME D'OCCASION ? MERCI

SECRETAIRE Du 26.09.88 A 14h18
PREMIERES IDEES SUR F32: Materiellement:
- 32 broches bus d'adresse
- 8 broches bus de donnes
- diverses broches signaux controle

Registres principaux 32 bits:
- T pour TOP: sommet pile data
- N pour NEXT: seconde donnee pile
- I index ou compteur repetition
- PC compteur programme
- OFS Offset compteur programme

liste non limitative

Operateurs:
- internes integralement 32 bits
- externes 8, 16 ou 32 bits

Operateurs:
- internes integralement 32 bits
- externes 8, 16 ou 32 bits

J'entend par operateurs internes, ceux qui effectuent exclusivement des operations registre a registre. Les operateurs externes sont ceux qui effectuent des operations memoire vers registre et inversement.
En limitant le format d'une instruction sur 8 bits, on dispose quand meme de 256 possibilites si l'on tient compte d'un projet RISC, donc amene a en compter nettement moins dans la realite.

Les codes instructions:
- registre vers memoire et inversement avec deux bits de poids faible indiquant taille des donnees a traiter. Il reste 6 bits de poids fort, soit 64 possibilites.
- registre a registre sur huit bits, soit 256 possibilites. Soustraire les combinaisons exploitees par reg<->mem.

Exemple, le chargement de T depuis la memoire sera execute par une instruction de type LIT:

```
n CONSTANT (lit,)
: /08BITS 0 OR ;
: /16BITS 1 OR ;
: /32BITS 2 OR ;
: clit, (lit,) /08BITS C, ;
: lit, (lit,) /16BITS , ;
: dlit, (lit,) /32BITS , , ;
```

A l'execution du code operateur de CLIT, le microprocesseur F32 incremente PC de 2 unites apres avoir charge l'octet de poids faible de T avec la valeur qui suit le code opera-teur; pour LIT, PC est incremente de 3 unites; pour DLIT, PC est incremente de 5 unites.
Ce principe est applicable aussi aux operateurs @ et !.
Pour les operateurs executant des operations reg<->reg, ils agiraient integralement sur 32 bits.
Exemple, l'inversion du contenu de T et N sera execute par swap, soit n le code operateur de swap:

```
: swap, n c, ;
```

Ainsi, une operation simple sera lue en memoire en un seul cycle machine. Pour les sauts, il n'y aurait que deux cate-gories:
- saut conditionnel
- saut inconditionnel

Un bit d'indicateur Z autorise ou n'autorise pas l'execution du saut.
On peut envisager trois sorte de sauts:
- saut court, operande 8 bits
- saut moyen, operande 16 bits
- saut long, operande 32 bits

La valeur du saut est relative. soit n le code operateur de branch:

```
: (branch,) n ;
: sbranch, (branch,) /08BITS c, ;
: branch, (branch,) /16BITS , ;
: lbranch (branch,) /32BITS , , ;
```

nota: , , peut etre remplace par d. l'assembleur-compilateur choisira automatiquement une des trois longueur de saut en fonction de la taille du branchement. Il est fort improbable d'avoir a utiliser le branchement long.
Le saut conditionnel aura comme mnemonique ?branch.
Appel de sous programme par call,: identique a branch, mais en adressage absolu.
Il y aura un code specifique ret, de retour de sous-programme et non un bit de retour comme pour le NC4016.
En multi-tache, chaque registre sera duplique en hard et un selecteur de tache permettra d'acceder a un des registres de tache. Soit n le nombre de taches possibles, il y aura n registres T, (T1, T2, T3...Tn-1, Tn), mais a chaque instant, on n'accedera qu'au registre T de la tache courante.
Voici pour une premiere ebauche des caracteristiques de F32. Le debat est ouvert:
- liste des instructions necessaires
- classe des instructions de base

A partir de ceci, definition du mecanisme F32:
- en fonction de sa destination
- ses possibilites
- configuration materielle.

Bon courage...

TICK Du 28.09.88 A 12h53
HELLO
YA T'IL DES PETITS MALINS QUI ON DEJA COMMANDE DES SOFTS EN DIRECT AUX USA VOIR LES PUBS DE BYTE ON ON PAYE PAR CARTE VISA. LES MALINS EN QUESTION SERAIENT TRES GENTILS DE ME DONNER TOUS LES TUYAUX DANS MA BAL.
UN dBASE 4 AU TIERS DU PRIX C'EST PAS NUL.
MERCI D'AVANCE
TICK

SECRETAIRE Du 28.09.88 A 16h38
CONCERNANT COMMANDE DE SOFT AUX USA: DIFFICILE DE RECLAMER SI NUMERO DE CARTE VISA MAL EMPLOYE. COMME LA POSTE NE FAIT PAS DE MANDATS INTERNATIONAUX FRANCE<->USA. DEMANDEZ A VOTRE BANQUIER D'EMETTRE UN CHEQUE DE BANQUE EN DOLLARS. BEAUCOUP LE FONT. VOUS EVITEREZ LES SURPRISES. ET LE PAIEMENT EST AINSI JOINT A LA COMMANDE.

SECRETAIRE Du 28.09.88 A 16h43
CONCERNANT dBASE 4 A MOINS DE 1000 FR. VALABLE SEULEMENT SI VOUS ECHANGEZ AVEC L'ORIGINAL DE DBASE III+ EN VOTRE POSSESSION. POUR CONNAITRE LES CONDITIONS EXACTES. S'ADRESSER A LA COMMANDE ELECTRONIQUE. ATTENTION. DBASE 4 NE SERA PAS DISPONIBLE AVANT NOVEMBRE 88.

SECRETAIRE Du 28.09.88 A 16h46
J'AI PRIS CONTACT AVEC HARRIS AU SUJET DU RTX2000. DES QUE J'AURAI DES DOCS. JE PONDRAI UN PAPIER SUR CETTE MERVEILLE. 1ERE INFO: LE KIT RTX2000 AVEC MEM ET CIRCUITS COUTE QUAND MEME 2000$. MAIS SI NOS CONTACTS AVES HARRIS SE CONFIRMENT. JE PENSE AVOIR DES PRIX. J'EN CONNAIS QUI NE VONT PAS REGRETTER LE MONTANT DE LEUR COTISATION ET D'AUTRES QUI VONT SE REABONNER...

XXXXX Du 29.09.88 A 15h42
CONCERNANT RTX2000 (BTX->TELETEL)
LE 22.06.88 LE GROUPE DE TRAVAIL RTX2000 S'EST CONSTITUE A MUNICH. COORDINATEUR DU GROUPE DE TRAVAIL HERR MAX DIEZ. HARRIS SEMICONDUCTOR GMBH POSTFACH 1232 D-8057 ECHING TEL: (19-49) 089/319 00 536
LE BUT DU GROUPE EST, A COURT TERME, LE DEVELOPPEMENT D'UNE PLATINE AVEC FORTH ET EPROM.

JMBK Du 01.10.88 A 23h30
ATMOS ET JASMIN-2 CHERCHENT CONTACTS EN F-83 PARIS ET REGION PARISIENNE ECHANGE DE LOGICIELS BAL JMBK.

JMCFORTH Du 03.10.88 A 16h20
VEND CAUSE DOUBLE EMPLOI:
VOL.8 FOTH DIMENSION (86/87)...150 F
VOL.9 FORTH DIMENSION (87/88)...150 F
DESIGNING & PROGRAMMING PERSONAL EXPERT SYSTEM (C.TOWNSEND & D.FEUCHT)..140 F
TOOLBOOK OF FORTH VOL.1 (DR.DOBB'S).170F
TOOLBOOK OF FORTH VOL.2 (DR.DOBB'S).220F
PRIX AUQUEL IL CONVIENT DE RAJOUTER LES FRAIS D'ENVOI PLUS LA TAXE DE CONTRE REMBOURSEMENT. TEL 94.22.10.36 POSTE 14 AUX HEURES BUREAUX (JM COURET)
ADRESSE J.M. COURET 9 RUE RACINE 83000 TOULON

HARRIS RTX2000 Du 03.10.88 A 18h58
AVANT PREMIERE: HARRIS ANNONCE UN SEMINAIRE A PARIS LE 18 NOVEMBRE 1988 POUR PRESENTER LE RTX2000. PROCESSEUR 16 BIT TEMPS REEL EN LANGUAGE FORTH. POUR TOUS RENSEIGNEMENTS: HARRIS 9 RUE FERNAND LEGER 91190 GIF YVETTE TEL 69077802

EUREKA Du 03.10.88 A 22h23
SALUTATIONS A TOUS. JE VOUS PROPOSE DE M'AIDER A RESOUDRE UN PROBLEME INTERESSANT: JE POSSEDE UN ARCHIMEDES TRES RAPIDE ET JE VOUDRAIS TURBO FORTH EN MODE NATIF (ECRIS EN ASSEMBLEUR A R M RISC) JE CHERCHE DES SOURCES POUR ECRIRE TOUT DEPUIS LE KERNEL JUSQU'EN HAUT TURBO F. DES AMATEURS? SI OUI, ME REPONDRE DANS MA BAL OU ME TELEPHONER AU 47 78 99 67 APRES 21H EN DEMANDANT MICHEL A + POUR D'AUTRES SUJETS

SECRETAIRE Du 04.10.88 A 10h43
J'ai commence a ecrire un meta-assembleur NOVIX NC4016. Depuis j'ai recu la doc du HARRIS RTX2000. Houla la! C'est vraiment une bombe! Pour info:
- registres:
SLR limite de pile
SPR pointeur de pile
IMR masque d'interruption
IBCR base/controle d'interrupt.
DPR data page adresse
UPR user page adresse
CPR code page adresse
UBAR user page adresse intra-segm
TCR0 | timers
TCR1 |
TCR2 |
TOP registre de sommet de pile
NEXT second element de pile
I index d'iteration
IR report sommet pile retour
MD registres transition pour
SR mutipl. et racine carree
PC compteur programme
CR registre indicateurs
Le RTX2000 peut lire des donnees en memoire selon le mecanisme INTEL ou MOTOROLA. Pour selectionner un de ces deux modes. il suffit de positionner le bit b3 du registre CR. A l'initialisation. c'est le mode MOTOROLA qui est selection-ne.
Le RTX2000 reprend la configuration materielle du NOVIX. a savoir:
- deux zones memoires independantes pour les piles de donnees et retour: taille 255 mots chaque.
- une zone memoire 64Kmot principale. selectable en plusieurs bancs additionnels. via le port x.
Mais surtout. il est tres performant sur le plan de la gestion des interruptions. Un debordement ou une insuffisance de donnees de pile donnees/retour genere une interruption.
En conclusion (sommaire...). le RTX2000 semble etre bien le micro-processeur ideal de developpement industriel applique a des applications type "temps-reel". Deja, son cousin NOVIX pouvait faire du traitement video avec ses 8MIPS (8 millions d'instr./sec).
Tres faible consommation electrique. maximum 1 watt (typiquement. 5mA/Mhz), temperature de fonctionnement: 5V+-10% a -40<T<+85.
Alors. mordus que vous etes de FORTH. a quand un C. PROLOG. FORTRAN. MSDOS. et tutti quanti ecrits en FORTH RTX2000.

SECRETAIRE Du 04.10.88 A 11h17
RECU DE FRED BEHRINGER. fonctions SQR:
1ere version:

```
 : SQR ( u1 --- u2)
   HERE ! 0 PAD !
   1 2 4 8 16 32 64 128
   8 0 DO DUP PAD @ + DUP * HERE @
         U> IF DROP 0 THEN PAD +!
       LOOP   PAD @ ;
```

2eme version de SQR. extraite manuel de ZECH.:

```
 : SQR ( u --- u)
   DUP 0= IF EXIT THEN
   16
   BEGIN   2DUP   0 SWAP UM/MOD SWAP DROP
           OVER +  2/
           DUP ROT   -   ABS   2/
    WHILE   REPEAT ;
```

3eme version. sur nombres 32 bits:

```
 : UD< ( ud1 ud2 -- fl)
   DUP 3 PICK =
   IF   ROT 2DROP U<
   ELSE SWAP DROP ROT DROP U< THEN ;
 : UD> ( ud1 ud2 --- fl)   2SWAP UD< ;
 : UD= ( ud1 ud2 --- fl)   D- D0= ;
 : UDMAX ( ud1 ud2 --- ud3)
   4DUP UD<   IF 2SWAP THEN 2DROP ;
 : SQR ( ud1 --- ud2)  HERE 2! 0 PAD !
  1 2 4 8 16 32 64 128 256 512
  1024 2048 4096 8192 16384 32768
  16 0  DO  DUP PAD @ + DUP UM* HERE 2@
        UD> IF DROP 0 THEN PAD +!   LOOP  PAD @ ;
```

4eme version. code machine extraite du manuel de ZECH.:

```
  CODE SQR ( u --- u )
  CX POP  08 # DI MOV   0 # AX MOV   AX DX MOV
  BEGIN  STC   DX RCL   CX SHL   AX RCL   CX SHL   AX RCL
         DX AX CMP    >=  IF    DX AX SUB   DX INC
         ELSE  DX DEC    THEN   DI DEC  0= UNTIL
  DX SAR   DX PUSH   NEXT END-CODE
```